ETH zürich

Department of Computer Science
Algorithms and Didactics

WebTigerJython 3

A Web-Based Python IDE Supporting Educational Robotics

Clemens Bachmann

Master's Thesis

2023

Supervisors: Prof. Dr. Dennis Komm
Alexandra Maximova

Abstract

As the importance of computer science in education has increased, so has the significance of high-quality teaching materials and programming tools. TigerJython and WebTigerJython have been designed as integrated development environments (IDEs) for programming in Python, with easy setup and usage. While WebTigerJython can be accessed through a web browser, TigerJython requires installation. However, TigerJython supports a wider range of features, including programming robotics, which was not possible in WebTigerJython due to the limitations of web apps. With advancements in web technology, we are confident that the entire functionality of TigerJython can be ported to a web app. This is what we are trying to accomplish with WebTigerJython 3.

In this thesis, we lay the groundwork for WebTigerJython 3, which is a rewrite of the WebTigerJython IDE. As in its predecessors, Python syntax has been extended with elements such as the repeat loop. Python can be executed directly in the browser without the need of a backend server.

The focus of this thesis was making the IDE usable for programming robotics. We began by providing support for programming the micro:bit, with plans to eventually support additional robots in the future. The IDE offers the ability to export code for the micro:bit and flash the robot directly from the browser. We have also included libraries for programming micro:bit extension boards, such as the Maqueen robot and an LED ring.

Additionally, we have included a reference containing a collection of frequently used commands, along with code snippets that can be easily inserted into the the code editor using drag and drop.

Acknowledgement

I would like to thank the following people, without whom I would not have been able to complete this thesis. Prof. Dr. Dennis Komm for giving me the opportunity to write my thesis in his group. My supervisor Alexandra Maximova for providing feedback and guidance in our weekly meetings, implementing some of the features and her administrative efforts such as organising a course and providing course material for to test the IDE. Cédric Donner, who helped me with the technical aspects of the project. Elizabeta Cavar and Edeltraud Falch, who let me test WebTigerJython 3 in their robotics course. Andreas Aeberli, who cooperated with me working on the same code base for his thesis. Sara Steiner, for proofreading and providing feedback to early draft versions of the thesis. Jarka Arnold, for providing the robotics wrapper libraries. Tobias Antensteiner, for helping me implement some features and Nicole Roth (formerly Trachsler) for providing usage data of WebTigerJython.

Contents

Chapter 1

Introduction

The Center for Computer Science Education at ETH Zurich ("Das Ausbildungs- und Beratungszentrum für Informatikunterricht der ETH Zürich", ABZ) supports schools and teachers in elementary, secondary, and high schools who want to build up or expand their computer science lessons accordingly. To support teachers holding computer science courses, the ABZ developed their own IDE: TigerJython.

The TigerJython [28] IDE has been used for many years in educational settings. One of the main goals was to design a programming environment that would be easy to set up for educators and support a wealth of educational tools. In the 2010s, TigerJython was perfectly suitable for this because it could run on all Windows, Mac, and Linux computers. It used Jython [7], a Python implementation that was written in Java, which runs on the JVM (Java Virtual Machine) that is already installed on most devices. Another upside of TigerJython is that it supports not only programming in Python, but also working with turtle graphics, robotics, GameGrid, GPanel, and databases out of the box.

1.1 Motivation

During the last few years, some problems have surfaced. While Python is already at version 3.11, Jython is still stuck at version 2.7. Due to Jython being developed by a small group of people, the gap between the latest Jython and Python versions may become even larger in the future. Additionally, there are many Python packages that are implemented in C, such as NumPy or SciPy. As mentioned, Jython is implemented in Java; thus these packages would need to be reimplemented in Java to work in TigerJython. Also, with the rise of tablets, two new operating systems (iOS and Android) have to be supported. Fortunately, a web app does work on all platforms.

In 2018, the first version of WebTigerJython [36] was implemented by Nicole Roth (formerly Trachsler). This was a first attempt to move some features to a web application. As of now, the turtle, GPanel, and databases work in WebTigerJython.

However, since the implementation of WebTigerJython new standards like WebUSB have widened the range of functionality that can be offered by a web-based application. Due to these new technologies, we feel confident that all the functionality of TigerJython, such as robotics, can now be ported to a web app. Robotics was actually one of the key missing features in WebTigerJython, therefore, we decided to make it the focus of this work.

We decided not to continue developing the original version of WebTigerJython, but to reimplement it from scratch. The main reason for this was that WebTigerJython used Skulpt [30] as Python runtime written in JavaScript. Using Skulpt was a good decision when WebTigerJython was developed; however, it is no longer state-of-the-art. While Skulpt is only able to execute a 2.6-ish version of Python, with Pyodide [35], an alternative web-based Python implementation, it is possible to run the current Python version (3.11) in the browser. A more detailed comparison between Pyodide and Skulpt can be found in Section 3. Because most of the functionality of WebTigerJython was implemented on top of Skulpt, switching to another Python implementation meant that most of it had to be reimplemented and could not be reused.

1.2 Related Research

In our related research section we consider similar IDEs and have a look at the research evaluating the usage of the micro:bit in classes.

IDEs

As a basis for our project, there were TigerJython [28] and WebTigerJython [36]. There are also other IDEs designed for educational purposes that are similar to our project.

Thonny [21] is another IDE for programming Python, which provides a powerful debugging tool to visualize code execution. Like TigerJython, Thonny is a desktop application.

The Python Online Editor [31] is a web-based IDE that executes Python code on a backend server. It is also capable of exporting robotics code in the backend.

A team at ETH has developed Code Expert [6], an online IDE for students to work on exercises and exams. The code execution in code expert is also done on a backend server. Besides Python, Code Expert can also execute code written in other programming languages.

The Python tutor [24] is a tool for executing Python code and visualizing this process. Same as the Python Online Editor, said execution is done on a backend server.

With the introduction of Python implementations running in the browser, IDEs that are able to run Python code in the frontend directly emerged. One of them is the aforementioned WebTigerJython. WebTigerJython is based on Skulpt [30], a Python implementation written in Javascript.

Here I also want to mention XLogoOnline [20, 26], an IDE with focus on programming turtle graphics. Turtle graphics can be programmed with block-based programming and also the TigerJython language. The TigerJython implementation of XLogoOnline is also based on Skulpt.

There are also Python IDEs that are able to run Python in the frontend, similar to WebTigerJython [36], such as Papyros [12], Basthon [2], and Futurecoder [25], which are all based on Pyodide. Papyros is mainly an online Python editor, while the Basthon project created a kernel that was then integrated into an editor and a Jupyter notebook. Futurecoder is also a Python editor extended with a Python course to learn programming from scratch. However, all of those IDEs have their main focus on executing Python in the browser, while we also want to program robots with it.

To program the micro:bit, there are some existing tools. For block-based programming, there is the web-based MakeCode editor [9] by Microsoft. The MakeCode editor also supports functionality to program robots such as the Maqueen or Maqueen Plus. The micro:bit can also be flashed with USB here.

The Scratch IDE [17] also supports block-based programming for the micro:bit. In contrast to the other editors, the micro:bit is not flashed. The user can establish a Bluetooth connection with the device. The code is then executed in the browser and the micro:bit is controlled with a Bluetooth interface.

There is an editor called Strype [39] that implements a frame-based approach to programming the micro:bit. Frame-based Programming is somewhere between the block- and text-based approaches. The Strype editor transpiles the code to Python to run it on the micro:bit.

There is also a web-based micro:bit Python editor [11] by the micro:bit Foundation. However, neither Strype nor the micro:bit Python editor have an extension for programming micro:bit extensions such as the Maqueen robot, which we implemented in our IDE.

micro:bit

We decided to work with the micro:bit [33] which has been developed by the BBC for computer science education. The first version of the single-board computer was released in 2016, and it contained a wide range of sensors.

In surveys J. Austin et al. [22] conducted they found out that most students said the micro:bit made computer science more interesting and also motivated girls choosing computing as a school subject.

In the work of Markus Tyrén [37], they investigated which technical pitfalls existed when designing teaching materials for the micro:bit. Some of the pitfalls were related to the IDE they were using. For example, they identified the issue that when running a course with iPads, one of the problems was that students without an app store account were unable to download the app. This issue will not occur when working with our web app, because no download is required. Students only need to open the app in their web browser. Thanks to the introduction of Progressive Web Apps, our app will be installable from the browser without the usage of an app store. However, we identified another pitfall which is that browsers on the iPad cannot flash the micro:bit.

1.3 Requirements

In this chapter, we take a closer look at the critical requirements for the base implementation of WebTigerJython 3 and the features it needs to be able to be extended to.

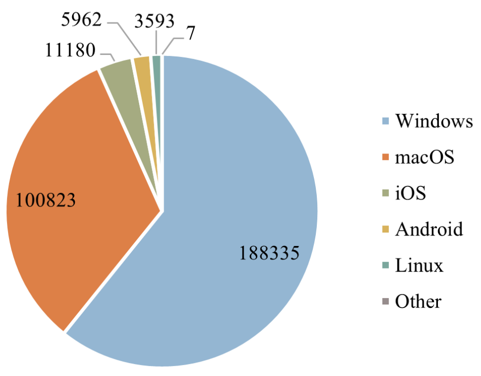


Figure 1.1. The number of WebTigerJython sessions by operating system in 2022

Simplicity

Many current IDEs are very powerful; however, they also have a very crowded user interface. We don't want to cause too much extraneous load on the student. So one of the main goals is to show the desired functionality, no more and no less.

Cross-Plattform Compatibility

When TigerJython was first implemented, it ran on the Java Virtual Machine (JVM). This decision was made due to the fact that the JVM was pre-installed on most devices. Therefore, setting it up could be done with one simple installation, and it worked on Linux, Mac, and Windows. With WebTigerJython, this was even simpler due to only having to open a website. It also made the application accessible for other operating systems (iOS, Android, etc.). In Figure 1.1, you can see the usage by operating system in 2022. Tablet usage is hard to assess exactly due to the fact that Windows tablets are not classified distinctly. Although even in a conservative estimate, it is a noteworthy amount. With the increasing usage of "bring your own device" initiatives in schools, this might increase even further. Therefore, cross-platform compatibility should be preserved and increased if possible.

Python

The application should run on current Python versions with all of its features. The standard Python syntax should be extended to the TigerJython language.

The TigerJython language is a superset of Python with some additional functionality. It specifically adds three features: the repeat loop, an improved input function, and some additional accessor functions for arrays.

In my work, I only implemented the repeat loop., it can be used without an argument. In that case `repeat:` is equivalent to `while True:`. It can also be used with an argument in which case it is equivalent to a for loop with the argument, specifying the number of iterations, without a control variable. For example, `repeat 4:` is equivalent to `for _ in range(4):`.

The improved input functionality replaces the regular Python `input()` function. In contrast to the regular input function, this function parses the input. If it is a number, it casts it as such. The regular Python input functionality handles any input as a string.

TigerJython also introduced additional accessors to access list elements. Additional to brackets, the first and last element of a list can be accessed as fields. For example, the last element of a list x can be accessed with `x.last`.

Robotics

One of the missing features in WebTigerJython, which was supported in the regular TigerJython, was robotics. The Python Online Editor has some limited robotics support. It is not able to flash robotics devices via USB but it can create a code export, which can then be manually loaded to the micro:bit. These code exports are created on the server, therefore a steady internet connection is required. With the new WebTigerJython 3, the goal is that robots can be flashed directly from the browser without any server interactions. As a first goal, we want to support the flashing of the micro:bit as a proof of concept. Further robots can be added in the future.

Potential Support for Further Libraries

One of the downsides of WebTigerJython was that its functionality was not as rich as that of the original TigerJython. The goal of WebTigerJython 3 is to be able to support all the functionality that the original TigerJython could. With WebTigerJython3 we lay the groundwork to allow adding this functionality in the future. These features are part of the TigerJython functionality, and if we want to have only one version of TigerJython in the long term, it is crucial to support these features.

Turtle Graphics Turtle graphics is one of the most popular ways to introduce programming to children. The simplicity and direct user feedback give the user immediate information about the execution state. Turtle graphics was already a part of TigerJython and WebTigerJython, and it should also be included in WebTigerJython 3. The development of turtle graphics is already in progress and is being implemented by Julia Bogdan as part of her Bachelor thesis.

GameGrid The first contact many children have with the computer is by playing games. Giving students the ability to create simple games by themselves is fun and can be used to introduce object-oriented programming. GameGrid was part of TigerJython and was built as a wrapper for JGameGrid, a Java library for programming games. Adapting this is not trivial and makes it necessary to reimplement the library in Python. Porting GameGrid is part of Andreas Aeberli's Master's thesis, and he has already achieved promising results.

GPanel TigerJython and WebTigerJython support a Graphics library called GPanel. It is used to draw shapes on a canvas, similar to turtle graphics. In contrast to turtle graphics, you can draw shapes from any position and not only the position of the turtle. Implementing GPanel might also be part of Julia Bogdan's thesis if time permits.

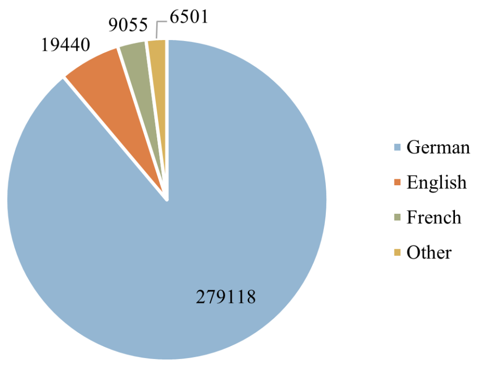


Figure 1.2. The number of WebTigerJython sessions by browser language in 2022

Database WebTigerJython should be able to be used to use databases. Python libraries such as SQLite3 should be accessible and also TigerJython libraries such as database1. Database1 is a library for teaching databases implemented for TigerJython. The library was adapted for WebTigerJython by Selin Barash [34].

Localisation

An additional requirement is that the application should be multilingual and it should be easy to add additional languages. As can be seen in Figure 1.2, the browser language of most users is German. However, we still have a non-negligible amount of users who use our tool in different languages.

Modularity

One further thing is that the application should be as modular as possible, due to the fact that multiple students will be adding further features to it. If possible, single modules should be replaceable and work independently of each other.

Chapter 2

Design of the IDE

In this chapter, we will be discussing the thoughts behind the decisions made for programming WebTigerJython 3. We will give an overview of how the different components are designed and how they interact with each other.

2.1 Design Decisions

In this section, we will talk about the design decisions made for the app infrastructure.

Application Type

As already mentioned, TigerJython was implemented as a native desktop app, while WebTigerJython was implemented as a web app. In this section, we will talk about the advantages and drawbacks of the two. We decided to implement WebTigerJython 3 as a web app, more specifically a Progressive Web App (PWA) [13].

Cross-Platform Compatibility One of the main drawbacks of native applications is that they are implemented for a specific platform. This means that, if we want to make a cross-platform native app, we'd have to implement the app for any platform. This would need a lot of resources for development and maintenance. Native apps have to be installed before usage with installation files or more and more commonly through app stores. Installing applications from an app store was one of the technical pitfalls mentioned in the work of of Markus Tyrén et al. [37] due to the user being required to have an account.

In contrast, web apps run on any platform without any installation required, besides a web browser of course. The introduction of PWAs made it possible to install web apps from the browser. When visiting a website that is also a PWA, most modern browsers (Chromium based browsers and Safari) give you the option of installing them. Installing a PWA creates an icon on the launcher or home screen, start menu, or launchpad. When opening the installed PWA, it is opened in a separate window outside of the browser, without the address bar. If desired, a PWA can still be exported to the respective app stores.

Performance Native apps were often praised as being more performant for two reasons. The first one is that native apps can run more efficiently because they can be optimized for the particular device's hardware. The second reason is that responsiveness of web apps suffers because whenever new content is loaded, it has to be retrieved from the web.

While the effect of the first drawback can lead to longer execution times, the second is the more pressing one. A slow internet connection could make an app basically unusable. Therefore this was also adressed with the introduction of PWAs. PWAs add a layer between the application and the web in the form of a service worker. When first visiting the website, the service worker loads all the files specified in a manifest into the cache. When the user triggers a web request, the service worker intercepts it and checks if the requested file is already cached. If it is, it can be provided and does not have to be loaded from the web. So, we have the possibility to load all files in advance and make the app usable in offline mode.

Native Features The biggest advantage of native apps is their ability to access the whole paraphernalia of device-specific features. Some of these device-specific features are accessible from the browser with APIs while others remain exclusive for native apps such as the devices local file system.

In our application we need access to the USB ports. These are needed to flash USB devices such as the micro:bit. This can be done using the WebUSB API [19]. Unfortunately the WebUSB API is only implemented in Chromium-based browsers. Currently, iOS devices don't support any browser based on Chromium, therefore functionality that requires WebUSB will not be accessible on those devices. This is specified in Apple's App Store Review Guidelines [1].

Frontend Framework

While WebTigerJython was built with vanilla HTML/CSS/JavaScript, we built WebTigerJython3 with a frontend framework. Frontend frameworks optimize rendering and therefore the performance of the app. They also help to better compartmentalize the code. Dividing up our code into smaller components makes it easier to reuse single code objects, test, and maintain.

We looked into Angular, React, Vue.js, and Svelte. All of these frameworks are suitable for the needs of this project. We decided on Vue.js as it is known as the most beginner-friendly option. This lowers the barrier for future students to start coding on the project and extend the app.

TypeScript

One additional difference from the original WebTigerJython is that we work with TypeScript instead of JavaScript. Typescript is a strongly typed programming language that builds on JavaScript. It helps the user understand the code better and helps catch errors before they occur at runtime, making the application less prone to typing errors.

Deployment

We decided to deploy the application on a regular basis to check if all the features can be integrated well. For that, we automatically deploy our code. We have two

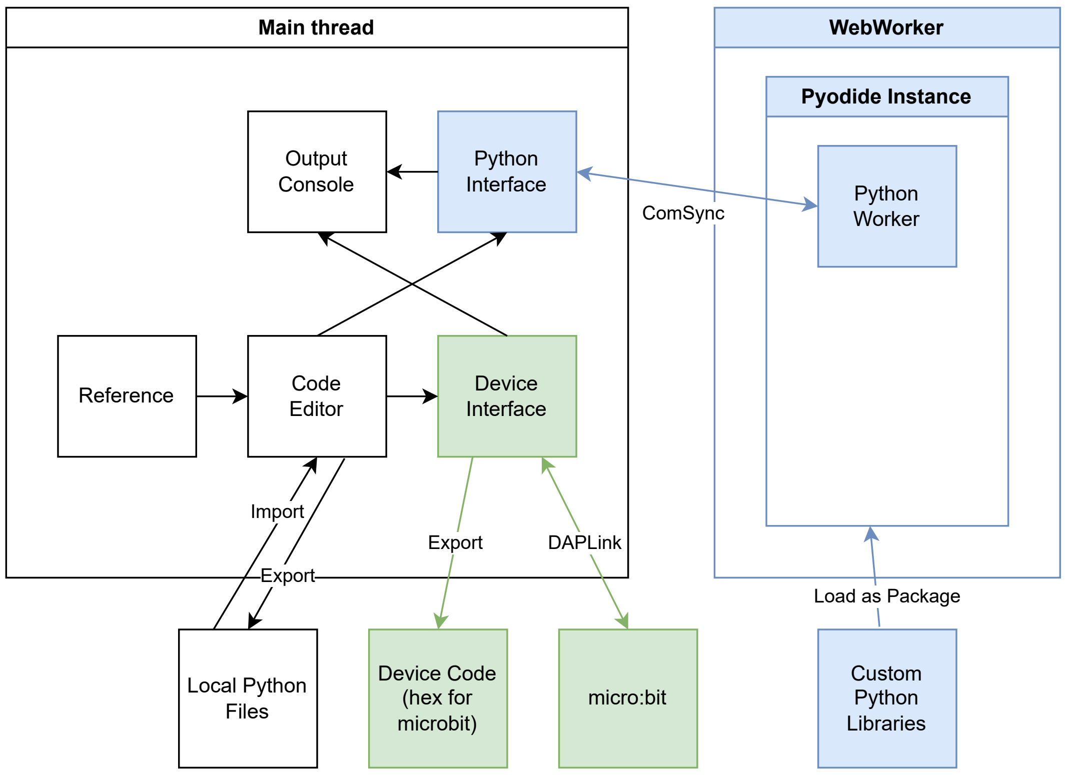


Figure 2.1. Abstract System Overview

branches in our Git repository, 'develop' and 'main', that are automatically deployed on new commits. This makes it easier to try out new features and see if they work in production.

2.2 Components

As seen in Figure 2.1, our system consists of two threads. The web worker thread loads and executes Python in the background, while the main thread handles everything else. How loading a Python runtime is done and how Python is executed is described in more detail in Chapter 3, while everything related to robotics will be described in Chapter 4. In this section, we will talk about all the other components and the graphical user interface.

Code Editor

To allow the students to write their code, we decided to use an already-implemented editor. The editor should help the students with programming without overwhelming them. We decided to work with the CodeMirror editor since it is very lightweight, powerful, and allows to write custom extensions easily.

Programming Language Support We added an extension to CodeMirror for language support. This was a module that was already offered by CodeMirror. We modified it by extending the language with further syntax, such as the repeat loop.

This extension contains Python code highlighting, auto indentation, and automatic closing of brackets. The language support also includes auto-complete for standard libraries. We decided to deactivate that in order to not overwhelm students with unnecessary information. In the future, this might be helpful as a toggleable feature.

Support for Drag and Drop Code Snippets In our reference component, we use code snippets that the user can insert into their code with drag and drop. We implemented an extension to the editor that allows to preview the content of the code snippets on drag and insert it on drop. The Micro:bit Python editor, from which we forked that feature, also used the CodeMirror editor. The reference is described in more detail in Section 2.2.

Comparison to Alternative Editors

There are other editors with similar functionality, such as the Ace editor, which was used in the first version of WebTigerJython, and the Monaco Editor. One big advantage of CodeMirror is that it supports the native selection and editing features of touch devices, which other editors do not. This is an issue that came up with the original WebTigerJython because there are schools that work with iPads or have a "bring your own device" policy, where parents often buy iPads or other tablets.

Canvas & Output Console

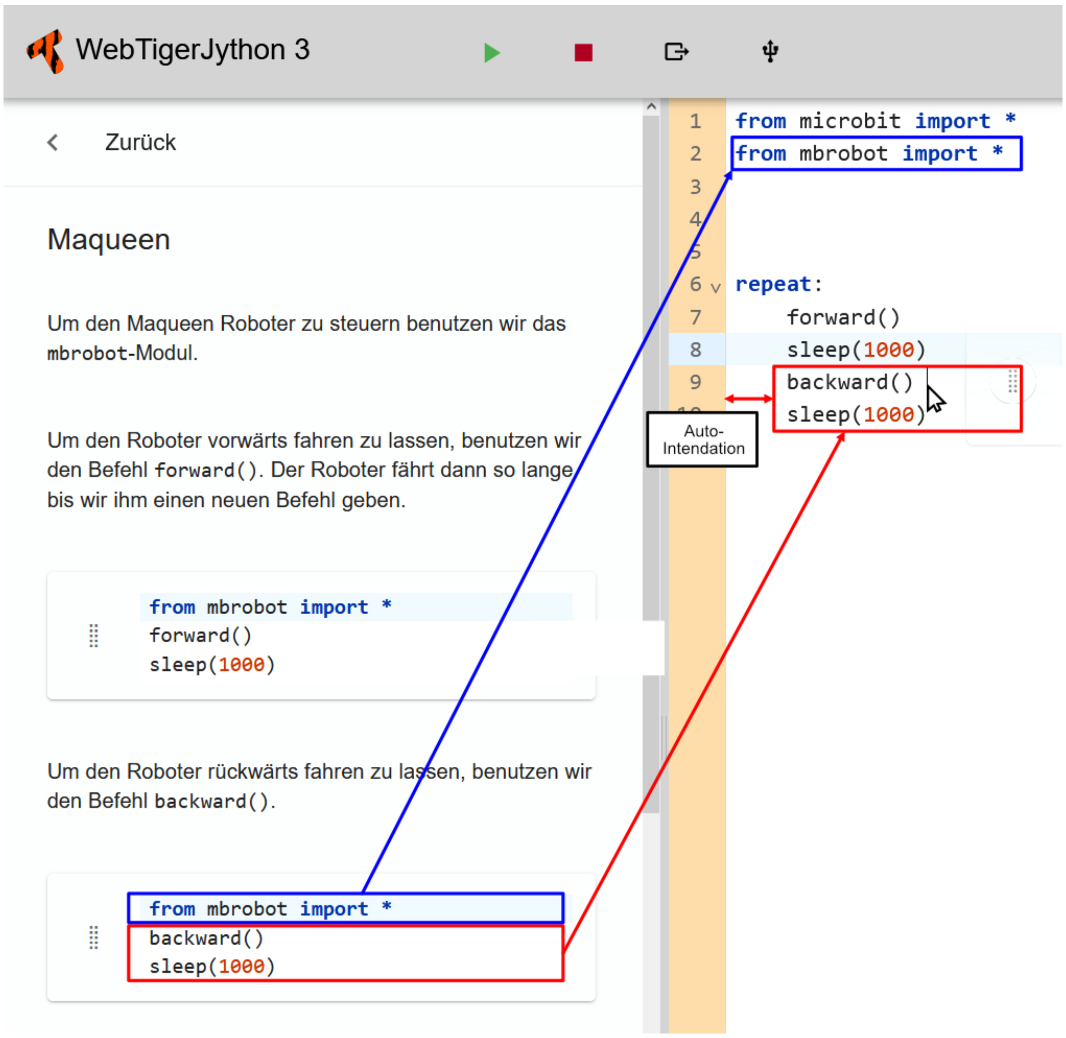
To display Python output, we created two areas. The lower right area in the IDE displays text output and errors. The upper area is used for graphical output. The layout is the same as it was in the original WebTigerJython. We did not implement any graphical output as part of this thesis and therefore, it is a placeholder for now. In the future, GameGrid, the turtle, and the graphical output of other libraries such as Matplotlib can be displayed there.

Reference

When learning to program, students often have to learn a lot of commands. To provide the students with an overview, we added a reference where commands can be looked up. In the reference, we explain the commands and add example code snippets. These code snippets can be dragged and directly inserted into the Python editor. Inserting code snippets with drag and drop should be intuitive, and it resembles working with code blocks. Many students start their programming journey with block editors such as MakeCode [9] or Scratch [17] before having their first coding experiences.

This idea is highly inspired by the micro:bit Python editor [11], from which we also forked some parts of our code. The way this is implemented is that we compute the position of the cursor when it is in the editor. According to that, we execute the changes that would be necessary to insert the code snippet at the cursor position. When the cursor position changes, we undo the previous changes and insert it in the new position.

The reference is organized in a multilayered folder structure where the commands are grouped by functionality.



Um den Maqueen Roboter zu steuern benutzen wir das `mbrobot`-Modul.

Um den Roboter vorwärts fahren zu lassen, benutzen wir den Befehl `forward()`. Der Roboter fährt dann so lange bis wir ihm einen neuen Befehl geben.

```
from mbrobot import *
forward()
sleep(1000)
```

Um den Roboter rückwärts fahren zu lassen, benutzen wir den Befehl `backward()`.

```
from mbrobot import *
backward()
sleep(1000)
```

```
from microbit import *
from mbrobot import *



repeat:
    forward()
    sleep(1000)
    backward()
    sleep(1000)
```

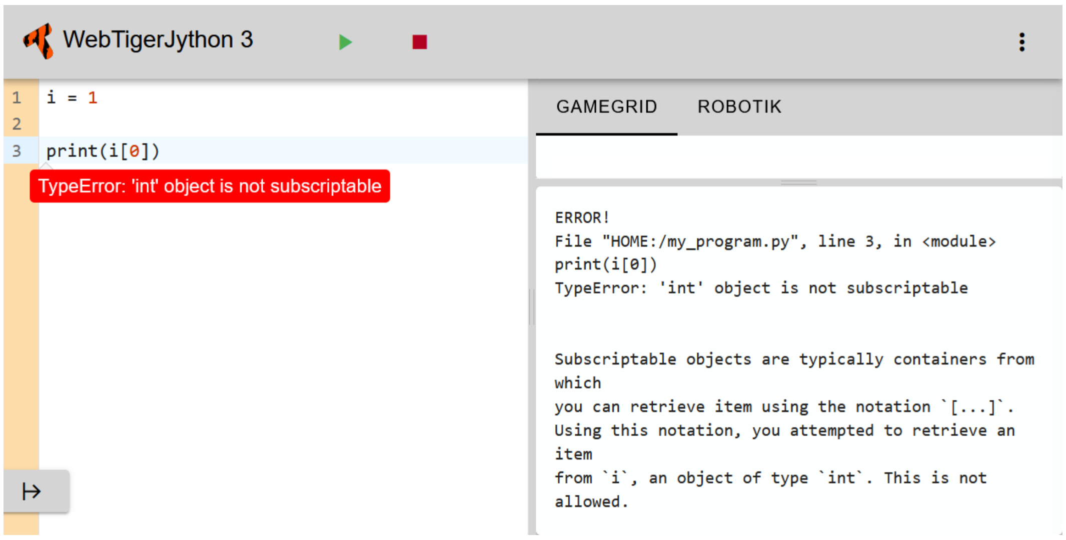
Figure 2.2. A visualization inserting code snippets from the reference. The code snippets can be inserted using drag and drop. The import is automatically inserted on top.

2.3 Error Messages

To provide the user with effective feedback during coding and assist them in recovering from errors, we have implemented multiple strategies.

Syntax Errors

To provide feedback for syntax errors, we use the TigerPythonParser [27], which was already used in TigerJython and WebTigerJython. Before executing or exporting the code, we check for syntax errors. If one is found, we prevent the code execution and display the error message directly in the code.



```
i = 1

print(i[0])
```

TypeError: 'int' object is not subscriptable

```
ERROR!
File "HOME:/my_program.py", line 3, in <module>
print(i[0])
TypeError: 'int' object is not subscriptable


Subscriptable objects are typically containers from
which
you can retrieve item using the notation `[...]`.
Using this notation, you attempted to retrieve an
item
from `i`, an object of type `int`. This is not
allowed.
```

Figure 2.3. The location of the error message is directly displayed in the code. The standard error message is displayed in the console output. In this example, Friendly Traceback is also added, a library that provides the user with further feedback.

Runtime Errors

For runtime errors, we give the user multiple levels of feedback. There is the possibility to just use the regular Python error message. We additionally displayed the error message within the code, as shown in Figure 2.3. They are sometimes hard to understand, especially for novice programmers, so we added the possibility to enhance the error message with a plain text explanation. This explanation is provided by a library called Friendly Traceback [32]. Friendly Traceback is also able to provide these explanations in multiple languages, which is helpful for many novice programmers who may not speak English very well.

It was requested by some teachers to also include the possibility of only displaying the location of the error message, which we have also done.

2.4 Graphical User Interface

The design of WebTigerJython 3 is inspired by TigerJython and the original WebTiger-Jython. To some degree, TigerJython is an established brand, and large changes would confuse the current user base. Therefore, we have decided not to make significant changes. However, knowing that it would also be used on devices with touch interaction in the future, We decided that some changes had to be made to ensure the app's usability. For the Graphical User Interface, we used a library called Vuetify. Vuetify is based on Material Design, which is a design language developed by Google and is used in many other sites and applications. This makes the app seem familiar to users while also being consistent with established industry standards. The Material Design language is flat and minimalistic, and the components behave like physical objects. We have animations that communicate the interaction to the user, and shadows help the user to focus on the interactive elements.

We have a top bar with the most-often-used functionality and an options drawer

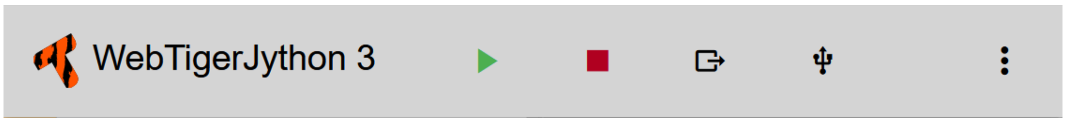


Figure 2.4. The top bar when a device is selected but not connected.

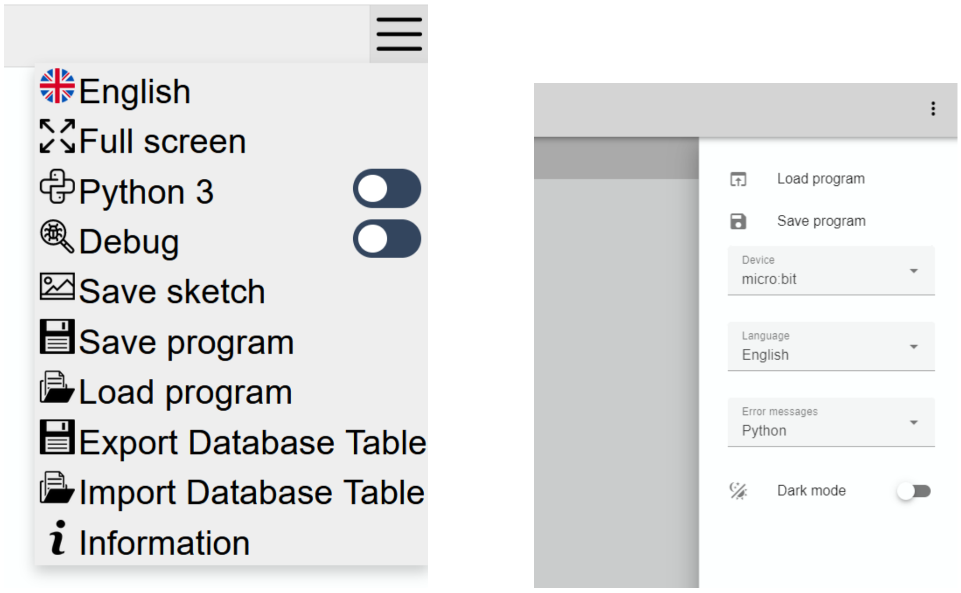


Figure 2.5. The old options bar compared to the new one.

on the right side with settings and less-often-used functionality. The main window is divided into three parts: the code editor, an area for graphical output, and an area for text output. Additionally, we added another drawer on the left side where you can access the reference with instructions for coding, containing code snippets that can be inserted into the editor using drag and drop.

Top Bar

The top bar normally just contains functionality to run and interrupt Python code. Additionally, there is the possibility to open the options drawer on the right. Further buttons can be displayed depending on the state of the application. If a device is selected (e.g., the micro:bit), further functionality such as export code is available. Depending on the connection status with the device, there is the ability to connect/disconnect and flash the device.

Options Bar

To fit the new design, we moved the settings to a drawer. When you open the drawer, the focus is set to it, and the rest of the application moves to the background and has a light shadow over it to communicate that to the user. We kept the small icons on the left side to make it easier for the user to find the right setting.

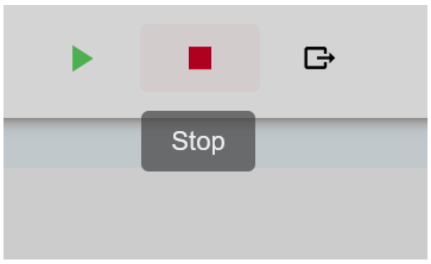


Figure 2.6. Icons and tooltips, the tooltip is displayed on hover

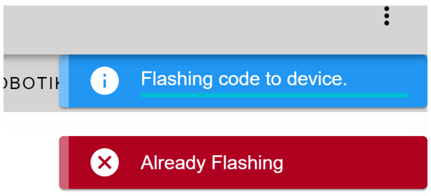


Figure 2.7. While the blue alert informs the user about a process in the background, the red alert communicates that something triggered by the user caused an error. In this example the error communicated that the user cannot flash while flashing is still in progress.

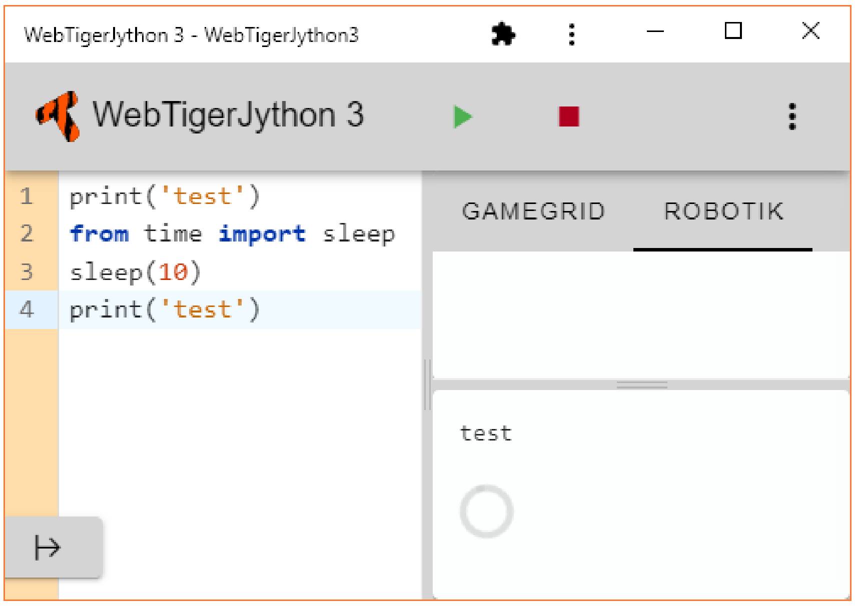
Icons & Tooltips

We decided to work a lot with icons due to the fact that they are easier to parse and understand in any language. Sometimes, what icons mean might be unclear to the user. If further information is needed, tooltips can be accessed by hovering over the icon. On touch, the same is achievable by pressing and holding the button.

Alerts and User Feedback

To make the user aware of what is happening, we use multiple methods of communication. One way of communicating with the user is to use alerts. We have four types of alerts, which are also color-coded. Error (red), success (green), info (blue), and warning (yellow) messages. Normally, the messages just stay for a few seconds and disappear. We also have messages with a loading bar to communicate to the user that an action is still being executed and to show the amount of progress that has been made.

Additionally, we use a circular loading icon to show the user that the code is still running, as seen in Figure 2.8



```
print('test')
from time import sleep
sleep(10)
print('test')
```

```
test
```

Figure 2.8. The circular loading icon communicates to the user that the code is still running

Chapter 3

Python Implementation

In the regular TigerJython, Jython is used to execute Python code. Jython is a Python implementation written in Java. The reason why Jython was picked is that the Java Virtual Machine was pre-installed on most desktop devices, and it could be distributed with a single JAR file, requiring no installation. An additional reason was that popular Java libraries used in teaching could still be used. The disadvantage of Jython is that the reference implementation of Python is written in C and developed by a large team, while the Java implementation is only developed by few and not that actively. Therefore, the supported Python version of Jython always lags behind, and this gap widens with time. At the time of publishing this thesis, the currently supported Python version in Jython is 2.7, while the current Python version in CPython is 3.11. Additionally, most Python packages are not supported in Jython since they are written in C and would need to be re-implemented.

In WebTigerJython, Skulpt was used as a Python implementation. Skulpt [30] is an in-browser implementation of Python written in JavaScript. Compared to Jython, there is an advantage that no Java Virtual Machine is required. It runs in any modern browser. Unfortunately, it has issues similar to those of Jython. The current version of Python that Skulpt supports is 2.6, and only Python libraries written entirely in Python are supported at all. Another drawback of Skulpt is that it is no longer actively developed, making it unsuitable for WebTigerJython 3.

There is an alternative web-based Python implementation written in JavaScript called Brython [18]. Brython is still receiving updates on a regular basis. However, the same issue that Jython and Skulpt have is also present: only pure Python libraries are supported.

There is also the approach of running the reference CPython implementation in the browser. While browsers do not directly run C code, it can be compiled to WebAssembly with the use of Emscripten [5]. WebAssembly is a binary code format that works in any current browser. Emscripten is a compiler toolchain which is able to compile C code to WebAssembly. Pyodide [35] is such a port of CPython to WebAssembly extended with a JavaScript interface. In Pyodide, only pure Python packages work by default. However, it is much easier to port CPython packages because they can also be ported using Emscripten.

We compared all solutions and concluded that working with Pyodide is the best strategy. The support for new Python versions and an overall better adaptability to libraries is a big plus, giving us more possibilities for the future.

	Python Version	Basis	Web	Adapting CPython libraries
Jython	2.7	Java	No	Hard
Skulpt	2.6	JavaScript	Yes	Hard
Brython	3.11	JavaScript	Yes	Hard
Pyodide	3.11	C / WebAssembly	Yes	Easy

Figure 3.1. A comparison between different Python implementations

3.1 Python Packages

Pyodide provides micropip, a lightweight package installer. Micropip in turn provides functionality to load Python packages directly from PyPI (Python Package Index). However, we have the limitation that we can only load packages that are written entirely in Python because the browser cannot run C code directly, in which many packages are written. Some of the Python packages written in C are ported by the Pyodide development team and can be loaded directly from Pyodide. There are also a few standard libraries that don't work in Pyodide and are therefore excluded. For example, Tkinter and libraries that depend on it, such as the native Python turtle [14].

In WebTigerJython 3, imports are checked and loaded before the execution. There is also the possibility of writing our own local packages, which we can then load into Pyodide.

3.2 Limitations

Pyodide still has some limitations. We run Pyodide in a web worker which runs in a separate thread. The web worker itself is single-threaded, which makes multithreading not possible at the moment. Running Python code in Pyodide is 3 to 5 times slower than running it with CPython. Running C code (in which the reference implementation is written) that is compiled to WebAssembly is only 2-2.5 times slower. This shows that there is room for improvement.

3.3 Non blocking execution

JavaScript and TypeScript are executed in a single thread, where blocking behavior can result in a frozen user interface. To prevent this, in the original WebTigerJython, asynchronous execution was used. With asynchronous execution, the code is still executed in the main thread, but not blocking it.

Another approach is the usage of web workers. A web worker is a separate, single-threaded script that is executed in the background. The web worker and the main thread do not share any memory.

The two communicate via messages. These messages take the form of JavaScript objects with the limitation that they have to be structurally clonable. When sending a message, the object sent is cloned and this clone of the object is sent to the respective other thread. Being only able to send clonable objects to the respective other thread brings the limitation that functions cannot be passed and therefore we cannot trigger functions in the respective other thread directly. To overcome this difficulty, we used a library called Comlink [23]. Comlink creates remote procedure call (RPC) proxys to expose functions to the respective other thread. This helps us initialize, run, or interrupt Pyodide from our main thread and also update our output console from the worker thread.

In our implementation, we always keep track of the state of the web worker in our main thread. That way, we can prevent the main thread from triggering tasks while the worker is still busy. An example of how this communication looks like can be found in Figure ??.

3.4 Interrupts, Input, and Sleep

To implement interrupts, input, and sleep, we utilized the pyodide-worker-runner [15] library, which simplifies the integration process. Details on how we implemented specific features are described in their respective section.

Interrupts

WebAssembly does not support preemptive multitasking, which is the basis of the Python interrupt system. So it is not possible to stop a running Python instance with JavaScript functionality. Therefore, the Pyodide team suggests using a SharedArrayBuffer that can be accessed from both inside and outside of Pyodide. The execution of Python can be interrupted by writing to the SharedArrayBuffer. We use pyodide-worker-runner to handle setting up the buffer and interrupts.

Input

Pyodide itself is not capable of handling Python's input functionality. To support input functionality, we override the Python input function with a callback function in our web worker. The callback function communicates with the main thread, which prompts the user for input. The input is then passed to the web worker, which then passes it to Pyodide. Pyodide is then able to resume execution using the user input.

Sleep

WebAssembly, in which Pyodide runs, does not support pausing execution for a specific duration of time. Here, we are overriding the Python `sleep()` function with a JavaScript callback function in the web worker. The callback function in the web worker is implemented by sending a request to a service worker, which then waits for a specified amount of time before notifying the web worker to resume Python execution.

3.5 TigerJython Syntax

As already mentioned the TigerJython language is a superset of Python with some additional functionality. It specifically adds three features: the repeat loop, improved input function, and some additional accessor functions for lists.

We implemented the repeat loop by parsing the code prior to execution or flashing. The loops are replaced with their corresponding standard Python equivalents (see

Section 1.3). We use regular expressions to identify the location of loops. Regular expressions (regex) are sequences of characters used to match patterns in text. In our initial implementation, we used the lookbehind [4] expression to identify sequences that were preceded by the “repeat” keyword. Unfortunately at time of implementation, the lookbehind expression was not supported in Safari browsers and caused the application to crash. It is a relatively new feature in regular expressions that was not supported by most browsers until a few years ago. Chrome first added it in 2017, while other browsers added it in the subsequent years. At the time of publishing this thesis, Safari supports this feature too. Because this parsing and replacement can also be performed without using this expression, we decided implement this feature without it. This will ensure that our application remains functional in older browsers that have not been updated yet.

The improved input functionality and the additional accessors are not yet part of WebTigerJython 3.

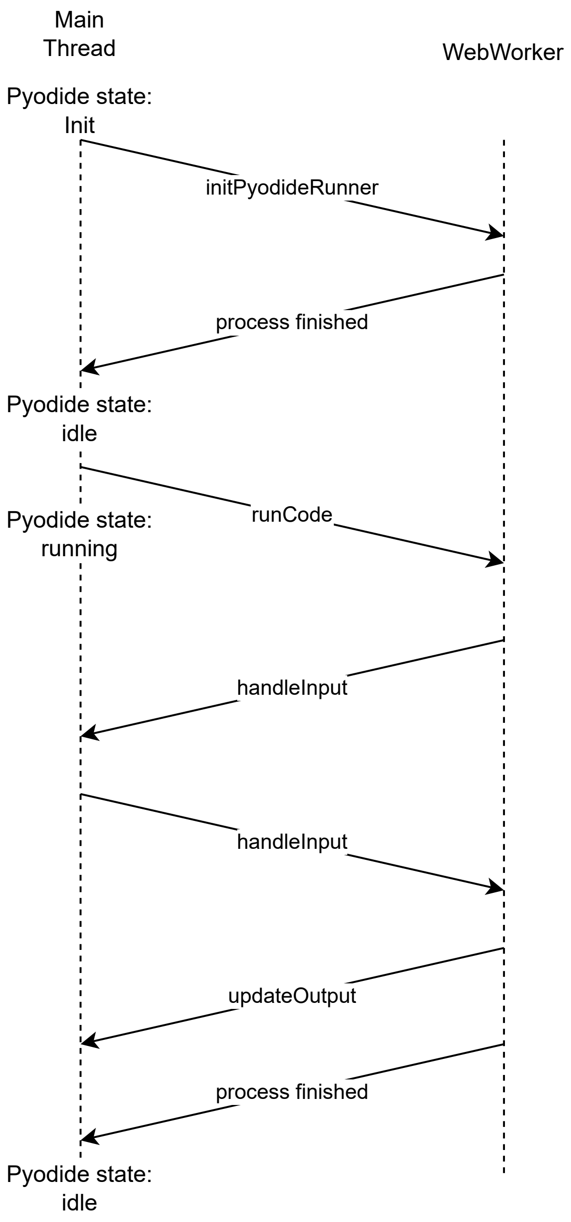


Figure 3.2. A visualization of how communication between main thread and web worker could look like.

Chapter 4

Robotics

One of the key features of TigerJython is to support programming for robotics. TigerJython supports multiple robots such as the Calliope Mini [3], the micro:bit [33], and the Lego Mindstorms EV3 [8].

The Lego Mindstorms EV3 is a programmable robot based on Lego blocks. It can be connected to a variety of sensors and motors. The micro:bit and Calliope Mini are very similar devices because the Calliope Mini was developed based on the micro:bit. They are extendable single-board computers equipped with a few sensors and methods to give feedback to the user.

We decided to focus on one of these and build WebTigerJython 3 in a way that makes it simple to add further robots. We decided not to focus on the Lego Mindstorms EV3 because the robot is already a few years old (released in 2013) and is relatively hard to get since it is no longer produced. As the Calliope Mini ran into multiple memory-related issues during the last few years we decided to go with the safest option: the micro:bit.

The original WebTigerJython did not support robotics. When WebTigerJython was developed, WebUSB had not yet been introduced. Therefore, flashing from the browser was not possible due to technical limitations.

With the Python Online Editor, there was an attempt to make it possible to program robots in the browser. The Python Online Editor, is able to convert Python code into a HEX file readable by the micro:bit. The creation of this HEX file is done on a server and not in the frontend, therefore an active internet connection is required for programming. The HEX file can then be manually transferred to the micro:bit.

In WebTigerJython 3, our goal is to create exportable code within the frontend so that no request to a server is needed. This allows the application to be used in an offline setting. Exporting files and then manually moving them to the micro:bit is cumbersome. Therefore, we wanted to be able to flash the micro:bit directly from the application by only clicking one button, like it is possible in TigerJython.

4.1 micro:bit

The micro:bit [33] is a single-board computer designed for educational purposes. It contains a (5×5)-LED matrix and a speaker to give the user feedback. It also contains some sensors such as buttons, a light sensor, a compass, a touch sensor, and

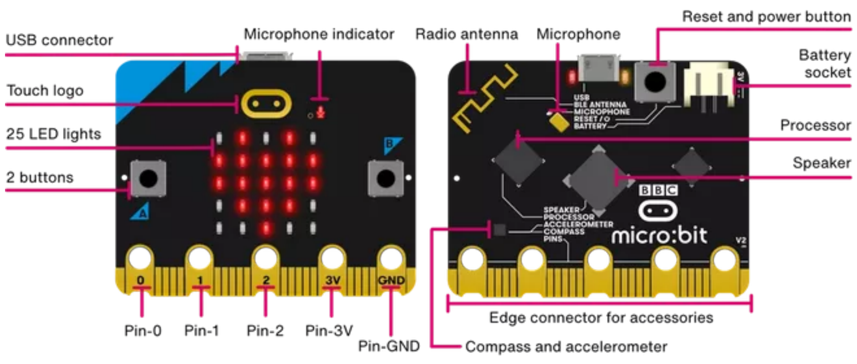


Figure 4.1. The micro:bit and its features [1]

a temperature sensor. Additionally, there are many modules available to extend the capabilities of the micro:bit, such as LED strips, wheels, and sensors like ultrasonic sensors. The micro:bit can be bought for a moderate price of about $30. It can be programmed with a large selection of programming languages. The micro:bit Foundation recommends two editors: the MakeCode micro:bit editor by Microsoft for programming the robot with blocks, or the micro:bit Python editor which they developed themselves for programming the robot with a text-based editor.

4.2 micro:bit Hex Files

In WebTigerJython 3, Python code for the micro:bit can be written. It is not possible to directly export Python code to the micro:bit because the micro:bit additionally needs a runtime environment that executes the code. There are multiple runtime environments supported by the micro:bit to run code written in different programming languages. Due to storage limitations, only one runtime can be installed at a time.

The micro:bit Foundation introduced a file structure [10] that combines a runtime with the given code, which can be loaded onto any micro:bit. This file is written in the Intel HEX format, a format often used for programming microcontrollers.

4.3 Flashing

These files can then be flashed onto the micro:bit. When the micro:bit is connected with USB it is displayed as a memory stick. When you drag and drop the HEX file on the micro:bit, the micro:bit overwrites its flash memory with the transferred HEX file. Flashing a file with drag and drop is cumbersome and also takes a lot of time, as seen in Figure 4.2, due to the fact that the whole MicroPython runtime environment, which is the majority of the code, has to be loaded onto the device every single time.

Another way of sending code to the micro:bit is by flashing it directly from the application. For that, we first set up a connection to our device using WebUSB. With DAPLink, we can establish a bridge between the browser and the micro:bit. Then, we can directly send our code from WebTigerJython 3 to the micro:bit. The flashing process compares the memory pages and only copies those that differ. If this fails, it

[1] https://microbit.org/get-started/user-guide/overview/

Partial Flashing with TigerJython	5s
Partial Flashing with WTJ3	2s
Full Flashing with TigerJython	60s
Full Flashing with WTJ3	30s
Copy hex-file using File Explorer on Windows	15s

Figure 4.2. A comparison between flashing times. We measured them on a computer running Windows 10. Flashing was done with USB 2.0, the current version supported by the micro:bit

falls back to a full flash, where the entire Python runtime environment is sent to the device. As you can see in Figure 4.2, a full direct flash takes longer than flashing by transferring a hex file. However, transferring a HEX file with drag and drop is more cumbersome. While testing the device, full flashing only occurred when another runtime was loaded, therefore, overall flashing was much faster. Another thing we have to mention here is that flashing with WebTigerJython 3 is substantially faster than with TigerJython. However, it should be noted that we did not implement the flashing process ourselves, we forked it from the micro:bit Python editor [11].

While direct flashing is faster, we have to mention it is dependent on the support of WebUSB. As of the publication of this thesis, only Chromium-based browsers support WebUSB. Most devices are able to run Chromium-based browsers, with the exception of iOS devices. There is a Chrome Browser app in the iOS App Store which is actually based on WebKit and not Chromium. It is a web engine and the only one that can be used on iOS devices. WebKit is the engine Safari is based on. WebKit and Firefox have yet refused to implement WebUSB as of the time of publication of this thesis due to privacy concerns [29,38].

There is also the possibility to flash the micro:bit using Bluetooth, and this is also supported by the MakeCode Editor [9]. Unfortunately, the micro:bit does not have enough memory to support both Bluetooth and the MicroPython runtime. Therefore, Bluetooth is disabled when the MicroPython runtime is flashed.

4.4 Extension Libraries

The micro:bit is extendable with multiple modules. It has multiple pins that can be connected to extension boards. We support the LED Ring, the Maqueen, and the Maqueen Plus extension boards. Programming these is not trivial and not suited for an educational setting, and therefore, wrappers have been implemented. We added multiple of those wrapper libraries to our IDE, and they can be imported like normal Python libraries. Before we export our code or flash it to the micro:bit, we check if these wrapper libraries are needed and add them accordingly. All of these libraries are provided by the TigerJython Group [16]. We support the following additional libraries:

mbled The `mbled` library is a wrapper for an LED ring with multicolored LEDs (Figure 4.3). The ring consists of 24 RGB LED lights, which can be accessed

[2] Image 1: Clemens Bachmann

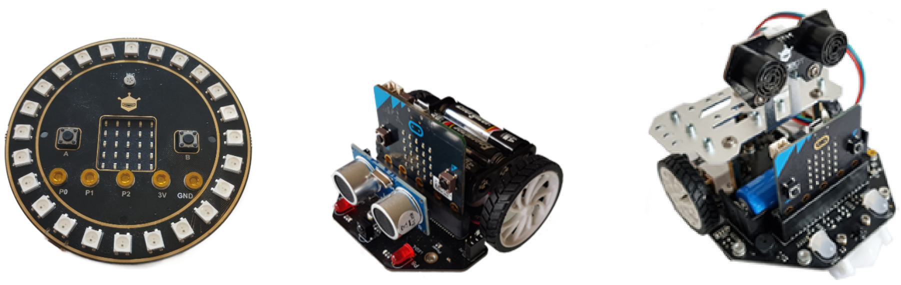
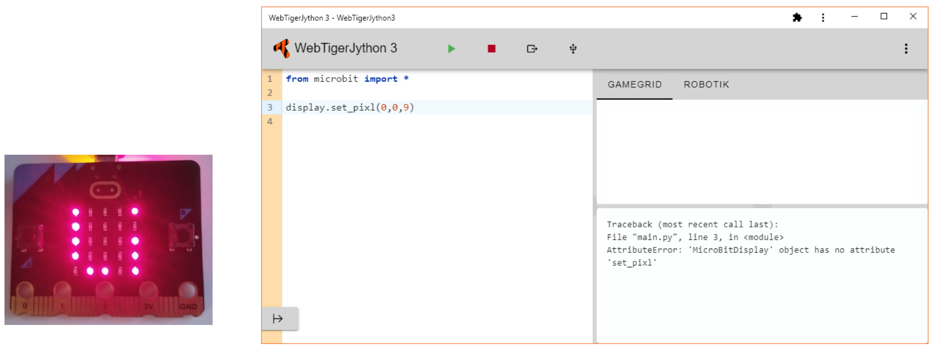


Figure 4.3. Three extensions for the micro:bit. An LED ring, the Maqueen robot, and the Maqueen Plus (left to right).[2]



```
from microbit import *

display.set_pixl(0,0,9)
```

```
Traceback (most recent call last):
File "main.py", line 3, in <module>
AttributeError: 'MicroBitDisplay' object has no attribute
'set_pixl'
```

Figure 4.4. Error messages are displayed on the micro:bit by scrolling through. The 'li' displayed on the micro:bit is part of the error message displayed in the IDE on the right.

separately. The wrapper consists of two functions: one to set the color of a single LED, and one to fill the entire LED ring.

mbrobot, mbrobotmot These libraries are implemented to control the Maqueen robot, as seen in Figure 4.3. The Maqueen robot has two wheels, some LEDs, an ultrasonic sensor to measure distances, and two line tracking sensors. The Maqueen can also be equipped with two additional servos to add more functionality, such as a shovel. The `mbrobot` library lets the user control the robot from a high level of abstraction. Functions such as `forward()`, `backward()`, `left()`, and `right()` can be directly used, and sensors can be read and servos can be controlled. The library "mbrobotmot" allows the programmer to access the servo motors of the wheels directly, enabling more complex steering maneuvers.

mbrobot_plus The `mbrobot_plus` library is similar to the regular `mbrobot` library. It also supports the additional features of the Maqueen Plus, which include additional LEDs and more line tracking sensors.

4.5 Serial Connection

Error messages are displayed on the micro:bit's (5×5)-LED matrix by scrolling through. This is done only once right after the error happens, and it is quite hard to read without missing any letters. The error messages can also be read through a serial connection established over WebUSB. In WebTigerJython 3, we read the errors from the serial connection and display them in the text output of the IDE. This connection only works while the micro:bit is connected with WebUSB.

Images 2 & 3: TigerJython Group https://python-online.ch/mbrobot.php?inhalt_links=mbrobot/navigation.inc.php&inhalt_mitte=mbrobot/mbrobot.inc.php

Chapter 5

Evaluation in Class

5.1 Setup

We tested our IDE during a semester course for primary school. The participants of the course were nine students between the ages of 9 and 11. The course was an optional module for high-performing pupils, and it comprised two consecutive lessons every week. Before the course, the pupils had already taken a computer science course for one semester, where they had learned to program with gturtle using WebTigerJython. In this course, the students learned to program the micro:bit with Python. The pupils used laptops provided by the school with Windows 10, and they used Microsoft Edge (which is based on Chromium) to access WebTigerJython 3. To program, they used the keyboard and the trackpad. The laptops were also equipped with a touchscreen, which some pupils used on a regular basis. I visited the course from the second to the sixth week to field-test the IDE and provide necessary help if any problems with the IDE occurred.

During the lessons, I watched the pupils and took notes. I also conducted structured interviews with all of the pupils. The interviews were conducted during the lessons, normally two every lesson, and took 5 to 10 minutes each. In one week, we did not conduct any interviews and postponed them to the following week. Furthermore, we tested the reference described in Section 2.2 in the final week during two consecutive lessons. At the end of those lessons, the pupils provided feedback in an online form consisting of five questions. During the the evaluation of the reference only five of the nine pupils were present.

5.2 Results

In this section we will talk about the qualitative feedback we received from students.

Overall Programming Environment

The pupils overall liked the new programming environment. When asked if they preferred WebTigerJython or WebTigerJython 3, most of them liked both IDEs equally, with a slight tendency towards a preference for the new IDE. When asked if they experienced bugs, pupils declined, except for a few pupils who experienced the double flashing bug in the first week, which is described in more detail in the

Subsection about flashing.

Error Messages

When asked which error messages they most often encountered, it was mostly import errors due to forgetting to import some functionality or typographical errors. Syntax errors were easily fixable due to the specific feedback received from the TigerPython Parser, and pupils stated that they were comfortable and confident in their ability to recover from them. However, runtime errors were not that easy to understand. The runtime errors with the micro:bit are displayed in the console. They are in English, which most pupils were not able to understand, and standard Python messages were hard to understand for the pupils as well.

Single File vs. Multiple Files

We asked the pupils how well they could manage their files. Currently, only one file can be opened. Files can be loaded or exported. I asked pupils if they thought it would help them if they could open multiple files at the same time. Most pupils did not see the need to open multiple files at the same time.

Robotics vs. Turtle Graphics

I also asked the pupils if it was hard to switch from turtle graphics to robotics. The pupils thought that, overall, the switch is not that hard. Having multiple libraries that have to be dynamically imported was perceived as more difficult. Nonetheless, the pupils reported being more enthusiastic about robotics due to the fact that there are more possibilities.

Flashing

I also asked if the process of flashing worked smoothly and if connecting to the device and then flashing was intuitive. This was mostly the case. At the beginning, we did not account for the fact that pupils could press the flashing button multiple times. If flashing is being executed while the IDE is still in the process of flashing, the file system will get corrupted, and a new full flash has to be executed. We therefore disabled flashing while being in the process of flashing and provided additional user feedback that the process of flashing is still being executed.

Reference

We let the pupils work with the reference for two hours and then questioned them. The pupils were enthusiastic about the reference. Usage varied a lot between pupils. Some pupils mentioned that the reference was really helpful for longer commands and that they did not have to search old worksheets to find the right commands. All Pupils stated that the speed at which they program would either stay the same or increase.

Other Observations

Pupils often pressed the “run” button and thought this would flash the code.

An additional observation we made was that when pupils produced very long text outputs, the text window would not scroll down automatically. Therefore, the pupils could only see the first few lines and they would have to manually scroll down to view the rest. This is rather annoying when you print out data, and you have to scroll down to see the most recent data.

5.3 Discussion

The observation that pupils slightly preferred WebTigerJython 3 over the original WebTigerJython could be attributed to the familiarity of the Material Design user interface. It could also be attributed to the fact that most pupils perceived programming robotics as more exciting.

Where most help was needed in recovering from error messages was when being confronted with runtime errors. Here, some additional help could really assist the students and improve independent learning. Specifically, with repect to translating or helping to interpret the error message.

Being able to open multiple files at the same time was not requested by the pupils. Possibly, this can be attributed to the exercises of the current course that could often be solved with a few lines of code. When projects get larger, working with multiple files becomes a necessity. Additionally, it has to be mentioned that most pupils did not get any experience working with an IDE that could open multiple files at the same time. If we add the feature of being able to open multiple files, we should make this feature toggleable not to overwhelm novice programmers.

For most pupils the switch to robotics was seamless. Pupils were able to gain abstract programming knowledge and apply it in a different context, which speaks in favor of a spiral curriculum.

The fact that the reference was only introduced in one week and adopted quickly was a positive surprise. Overall, the idea seems to be promising and should be further developed. We should keep in mind that in our class evaluation, we only worked with a small scope of functionality. If the scope of the reference increases, it should still be manageable.

Another aspect we should keep in mind is how we communicate functionality to the user. Many pupils used the “run” button, which executes the Python code locally in Pyodide, instead of the “flash” button. We should improve the intuitiveness of the user interface so that these mix-ups don’t happen anymore.

To address the issue that only the first few lines of output were displayed, and students had to manually scroll down to see more recent text output, we made the window automatically scroll down when new output appeared on the console. This enhancement was tested by the pupils the following weeks and appreciated.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In our thesis we managed to create a Python Web IDE. The IDE is able to run the latest Python Version and works in any modern browser.

The IDE also supports code highlighting, auto indentation, auto closing of brackets. The TigerJython-specific repeat loop is also supported. Error messages are enhanced with FriendlyTraceback [32] and the TigerPythonParser [27].

WebTigerJython is implemented as a Progressive Web App that can be installed and used like a regular application.

Porting robotics to the web was possible thanks to WebUSB. With the introduction of many WebAPIs almost all functionality of TigerJython can be ported to the web. Here we have to keep an eye on support. Some of the APIs are not supported by certain devices, e.g., iOS devices do not support WebUSB. We should look for workarounds in the future to make all the functionality of WebTigerJython 3 usable on as many platforms as possible. Specifically, it will be important to focus on iOS devices due to their high market share in tablets.

During the implementation, we stayed confident that it should be technically possible to port all TigerJython features to WebTigerJython 3. Some of the educational libraries used in TigerJython that are written in Java need to be rewritten and this might not be trivial, but it should be feasible.

Being able to use a current CPython in the browser opens a lot of new opportunities. Now that libraries like Numpy, Scipy, and Matplotlib are usable, we could extend the scope of potential users to more advanced programmers.

While testing our tool in an educational setting we only experienced very few bugs and the feedback we got from the pupils and the teacher was very positive. Still a lot of further testing has to be done. However we feel confident that WebTigerJython 3 can replace the old WebTigerJython and potentially the regular TigerJython in the future.

6.2 Future Work

As demonstrated in the thesis, the IDE has already been tested in an educational setting and with some further refinement, it should be suitable for use in classrooms. In this section, we will explore the features that need to be added to make it a complete successor of the current TigerJython versions, as well as other features that could be of interest.

Further TigerJython Functionality

If we want to cover the full TigerJython functionality, all of the following tools have to be added.

Turtle Graphics The turtle is a tool often used to start programming. It is a cornerstone of the spiral curiculum developed by the ABZ. Therefore this is one of the most important features to be added to WebTigerJython 3 in the future. There are other projects such as Basthon [2] that already have implemented turtle graphics, so this should be feasible. The turtle will be implemented as part of the Bachelor thesis of Julia Bogdan.

GPanel GPanel is a tool designed to give the user the ability to draw on a canvas with simple commands. In contrast to turtle graphics, the user can draw shapes from any position, while in turtle graphics, he is only able to draw at the turtles position.

GameGrid During the work on my thesis, another student, Andreas Aeberli, has been working on an implementation of GameGrid that works on the web. GameGrid in TigerJython heavily relies on JGameGrid, a Java library. Therefore it is not trivial, but still feasible, at least to a large extent.

Debugger One of the most important things in an educational IDE is a good debugger. A debugger is a feature implemented in WebTigerJython and critical for WebTigerJython 3 to supersede the current version. Tobias Antensteiner is in the progress of implementing a debugger that can be integrated in WebTigerJython 3.

More Devices While TigerJython supports programming the Lego Mindstorms EV3, the micro:bit, the Calliope Mini, an ESP32, oxocards, and the Raspberry Pi, WebTigerJython 3 so far only supports programming the micro:bit. Some but not all of these devices will have to be supported by WebTigerJython 3. Additionally, we should keep an eye on which Python-programmable robots are popular and support them if possible.

Server-Side File Management

Files can be imported and exported. This is the same functionality provided as in the regular WebTigerJython. Giving students the possibility to create an account and store the files on a server would give them the possibility to continue working on their project on any device. Otherwise they would always have to keep their files with them. This could also make it easier to share code with their peers.

Multi-file Projects

When students work on more complex projects, organizing their code in multiple files can be helpful. This feature will become critical for older and more experienced students.

Reference

The reference is still in a proof-of-concept state. There remain some things that have to be implemented such as multi-language support. Additionally, if the reference keeps growing, we should try to make it more concise and the desired functionality easy to find.

Runtime Errors in Robotics

The runtime errors in robotics are hard to interpret, since they are only provided in English, which constitutes a language barrier for many students. Additionally parsing them is hard for novice programmers. Showing them in the editor, translating them or parsing them in advance would surely help the students.

Bibliography

[1] Apple App Store Review Guidelines. https://developer.apple.com/app-store/review/guidelines/. Last accessed: 09.05.2023.

[2] Basthon. https://basthon.sct.pf/. Last accessed: 23.04.2023.

[3] Calliope mini, A microcontroller with many possibilities! https://calliope.cc/. Last accessed: 30.04.2023.

[4] Can I use? - Lookbehind in JS regular expressions. https://caniuse.com/js-regexp-lookbehind. Last accessed: 28.08.2023.

[5] Emscripten. https://emscripten.org/. Last accessed: 28.08.2023.

[6] ETH Code Expert. https://expert.ethz.ch/. Last accessed: 13.05.2023.

[7] Jython. https://www.jython.org/. Last accessed: 08.05.2023.

[8] Lego Mindstorms EV3. https://education.lego.com/en-us/downloads/mindstorms-ev3/. Last accessed: 09.05.2023.

[9] MakeCode. https://www.microsoft.com/en-us/makecode. Last accessed: 20.04.2023.

[10] micro:bit, .HEX file format. https://tech.microbit.org/software/hex-format/. Last accessed: 30.04.2023.

[11] micro:bit Python Editor. https://python.microbit.org. Last accessed: 23.04.2023.

[12] papyros. https://github.com/dodona-edu/papyros. Last accessed: 23.04.2023.

[13] Progressive Web Apps, mdn web docs. https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps. Last accessed: 28.04.2023.

[14] Pyodide Python compatibility. https://pyodide.org/en/stable/usage/wasm-constraints.html. Last accessed: 09.05.2023.

[15] pyodide-worker-runner. https://github.com/alexmojaki/pyodide-worker-runner. Last accessed: 15.05.2023.

[16] ROBOTIK und IoT mit MBROBOT und MAQUEEN PLUS.

https://www.tigerjython4kids.ch/index.php?inhalt_links=robotik/navigation.inc.php&inhalt_mitte=robotik/mbrobot/mbrobot.inc.php. Last accessed: 09.05.2023.

[17] Scratch. https://scratch.mit.edu/. Last accessed: 20.04.2023.

[18] Skulpt, A Javascript implementation of Python in the browser for education. https://skulpt.org/. Last accessed: 28.04.2023.

[19] WebUSB API. https://wicg.github.io/webusb/. Last accessed: 02.05.2023.

[20] XLogoOnline. https://xlogo.inf.ethz.ch/. Last accessed: 13.05.2023.

[21] 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.

[22] Austin, J., Baker, H., Ball, T., Devine, J., Finney, J., De Halleux, P., Hodges, S., Moskal, M., and Stockdale, G. The BBC micro:bit: from the UK to the world. *Communications of the ACM 63*, 3 (2020), 62–69.

[23] East, D. Simplify Web Worker code with Comlink. https://davidea.st/articles/comlink-simple-web-worker/. Last accessed: 02.05.2023.

[24] 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, p. 579–584.

[25] Hall, A. Futurecoder. https://futurecoder.io/. Last accessed: 23.04.2023.

[26] 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.

[27] Kohn, T. Tobias-Kohn/tigerpython-parser: Enhanced error recognition in Python. https://github.com/Tobias-Kohn/TigerPython-Parser.

[28] 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, p. 1054–1060.

[29] Mozilla. Specification positions. https://mozilla.github.io/standards-positions/#webusb. Last accessed: 13.04.2023.

[30] Nijburg, A.-J. Skulpt, A Javascript implementation of Python in the browser for education. https://skulpt.org/. Last accessed: 28.04.2023.

[31] Plüss, A., and Arnold, J. Python Online Editor. https://python-online.ch/pyonline/PyOnline.php.

[32] Roberge, A. Friendly Traceback. https://friendly-traceback.github.io/docs/index.html. Last accessed: 20.04.2023.

[33] Schmidt, A. Increasing Computer Literacy with the BBC micro:bit. *IEEE Pervasive Computing 15*, 2 (2016), 5–7.

[34] Selin Barash, Prof. Dr. Dennis Komm, A. M. A Database Library for WebTigerJython, 2022.

[35] The Pyodide development team. pyodide/pyodide. https://doi.org/10.5281/zenodo.7570138, Jan. 2023.

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

[37] 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, p. 81–86.

[38] WebKit. Tracking Prevention in WebKit. https://webkit.org/tracking-prevention/#anti-fingerprinting. Last accessed: 13.04.2023.

[39] 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, p. 585–586.

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Eigenständigkeitserklärung

Die unterzeichnete Eigenständigkeitserklärung ist Bestandteil jeder während des Studiums verfassten Semester-, Bachelor- und Master-Arbeit oder anderen Abschlussarbeit (auch der jeweils elektronischen Version).

Die Dozentinnen und Dozenten können auch für andere bei ihnen verfasste schriftliche Arbeiten eine Eigenständigkeitserklärung verlangen.

Ich bestätige, die vorliegende Arbeit selbständig und in eigenen Worten verfasst zu haben. Davon ausgenommen sind sprachliche und inhaltliche Korrekturvorschläge durch die Betreuer und Betreuerinnen der Arbeit.

Titel der Arbeit (in Druckschrift):

WebTigerJython 3 - A Web-Based Python IDE Supporting Educational Robotics

Verfasst von (in Druckschrift):
Bei Gruppenarbeiten sind die Namen aller Verfasserinnen und Verfasser erforderlich.

Name(n):	**Vorname(n):**
Bachmann	Clemens

Ich bestätige mit meiner Unterschrift:
- Ich habe keine im Merkblatt „Zitier-Knigge“ beschriebene Form des Plagiats begangen.
- Ich habe alle Methoden, Daten und Arbeitsabläufe wahrheitsgetreu dokumentiert.
- Ich habe keine Daten manipuliert.
- Ich habe alle Personen erwähnt, welche die Arbeit wesentlich unterstützt haben.

Ich nehme zur Kenntnis, dass die Arbeit mit elektronischen Hilfsmitteln auf Plagiate überprüft werden kann.

Ort, Datum	**Unterschrift(en)**
15.5.2023	[illegible]

Bei Gruppenarbeiten sind die Namen aller Verfasserinnen und Verfasser erforderlich. Durch die Unterschriften bürgen sie gemeinsam für den gesamten Inhalt dieser schriftlichen Arbeit.