System Description Documentation

For

Micro:bit Team 12's Editor

Created in Hilary/Trinity term 2021

Oxford University

Contents

Overview

Our project specification is to create a child-friendly web-based Python editor, to be used in conjunction with the BBC Micro:bit. The target audience for our product is tweens and teenagers, who are new at learning how to do text-based coding. They may or may not have previous experience with block-based coding. Therefore, every feature in our editor must be accessible to young people, and the editor must look playful rather than intimidating. Furthermore, our target users may have minimal IT (Information Technology) skills and may not understand how to download and move files. Therefore, our editor will communicate with the Micro:bit over WebSerial, so that the student will be able to directly flash their code to the attached device without needing to first download their code. The project must be entirely front-end based and must compile in the browser.

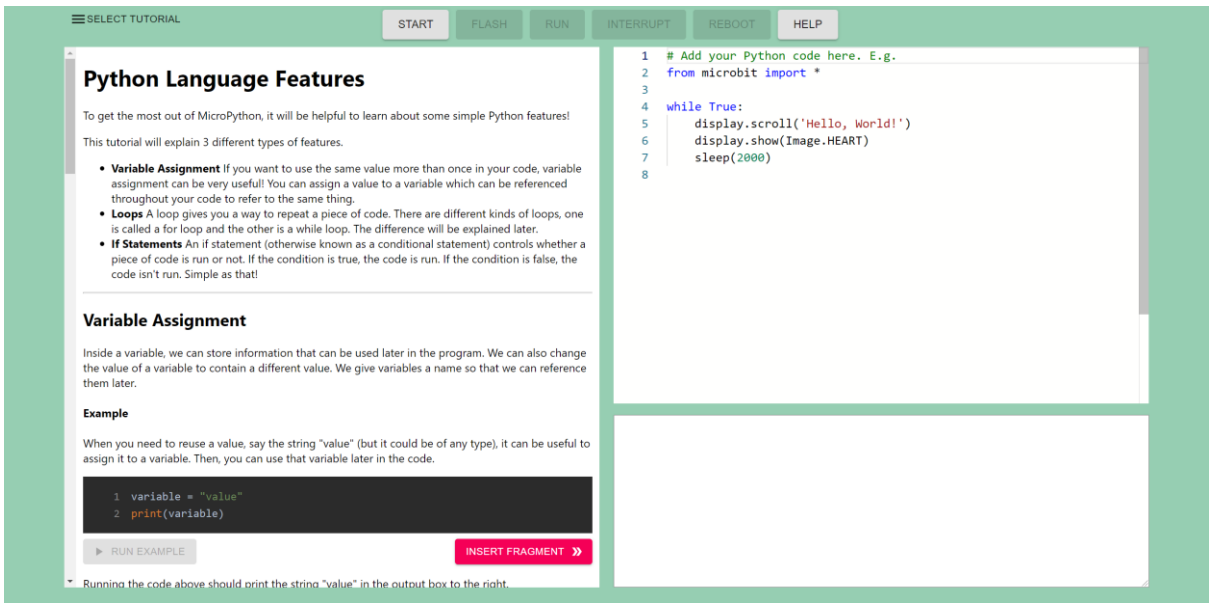
Features that the customer would like to see include:

- A Jupyter-like tutorial experience, which allows narrative and code to be mixed.
- A “Rubber duck” debugging chatbot, that takes the user through a flowchart to solve their problem.
- Ability to communicate to and from the Micro:bit over WebSerial.
- An autocomplete function for the user’s code.

Usage

Starting Out

When the user first opens the app, they should see something like this.



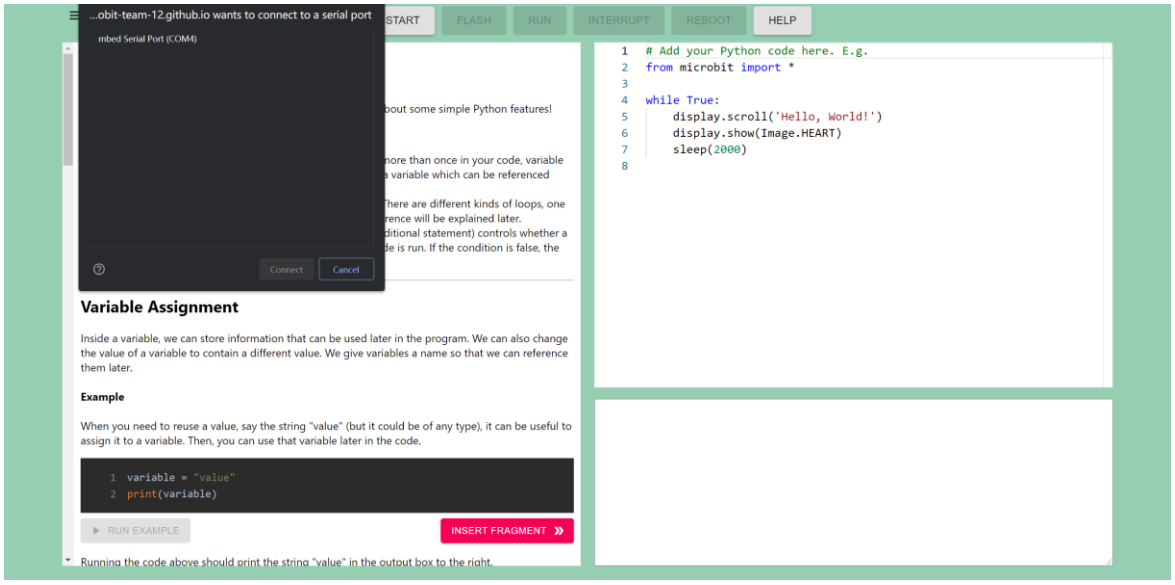
On the left, there is the tutorial window (to start, this will be the ‘Python Language Features’ tutorial as shown). In the bottom-right there is the code output window and above it is the code editor window, which will already contain some starting code.

At the top-left, there is the SELECT TUTORIAL button, which allows the user to switch tutorials. Finally, there are six buttons at the top: START, FLASH, RUN, INTERRUPT, REBOOT and HELP.

Connecting to the Micro:bit

To connect the Micro:bit to the app, the user will first need to plug it into their laptop or pc using a USB to micro-USB cable.

For initial connection, you will need to select serial port of Micro:Bit in the browser permission window. But for later connection, they simply need to click START button.



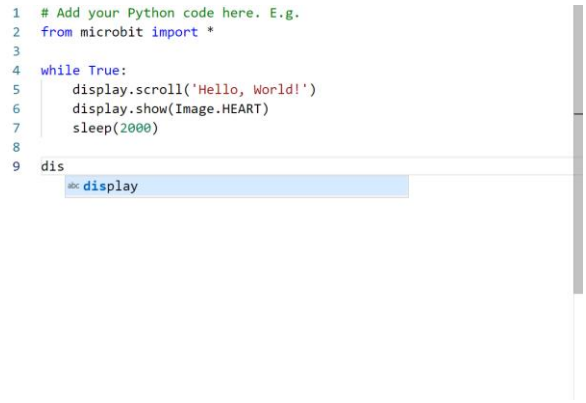
The Editor

The editor starts out with some initial code, like this:

```
1  # Add your Python code here. E.g.
2  from microbit import *
3
4  while True:
5      display.scroll('Hello, World!')
6      display.show(Image.HEART)
7      sleep(2000)
8
```

But they can type any Python code they want!

The editor has auto-complete functionality, to help beginners get to grips with both Python and the BBC Micro:bit (as well as saving time for non-beginners). The user can use the suggested completion by pressing enter or tab. For example, if they are trying to write 'display' or finding functions of library display:

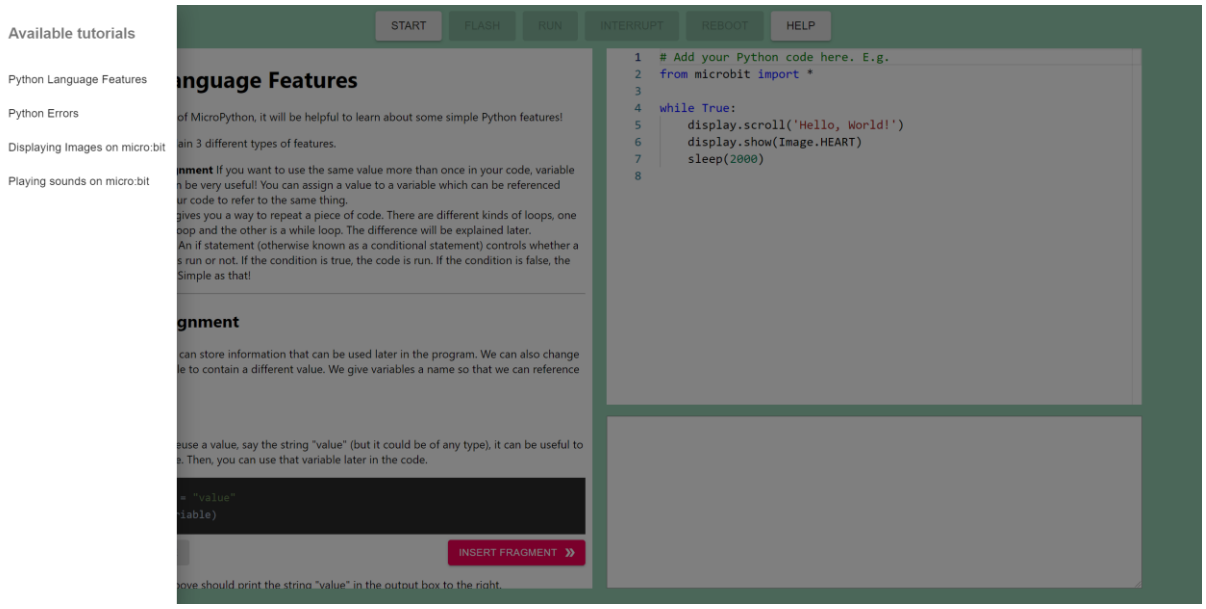


Running Code

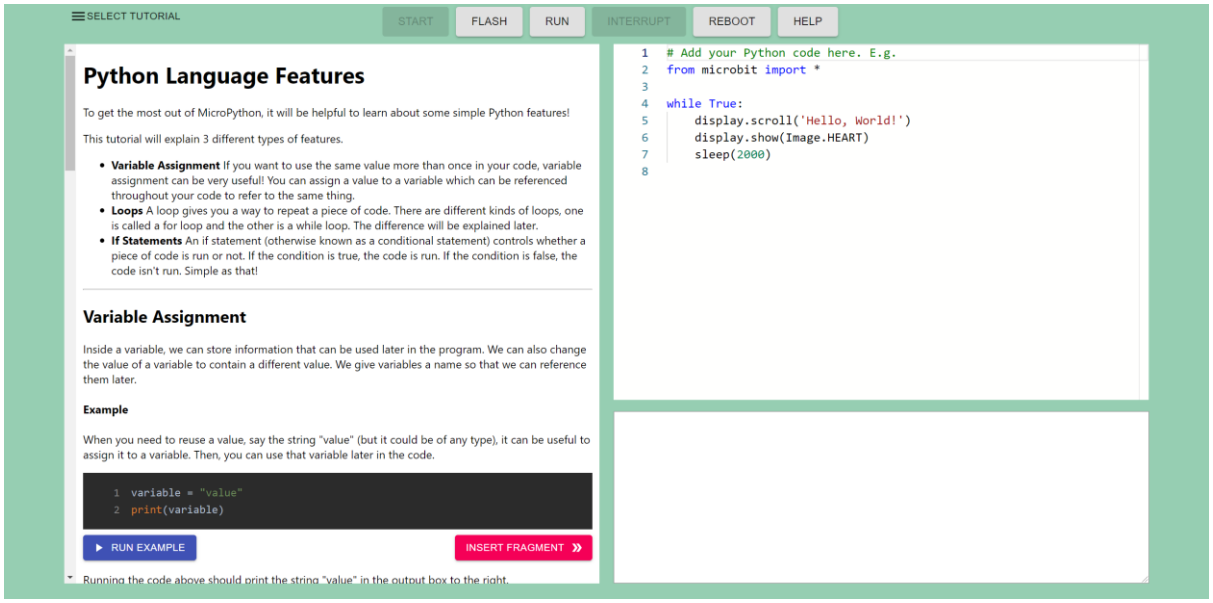
There are two buttons that cause code to be run on the Micro:bit - FLASH and RUN. Pressing FLASH causes the code to be flashed permanently onto the Micro:bit, such that it will persist if the Micro:bit is disconnected and then plugged back into a different power supply. On the other hand, RUN is useful for debugging, as it causes the code to be run on the Micro:bit over WebSerial. This means that the code stored on the Micro:bit is not changed when the user presses RUN.

Tutorials

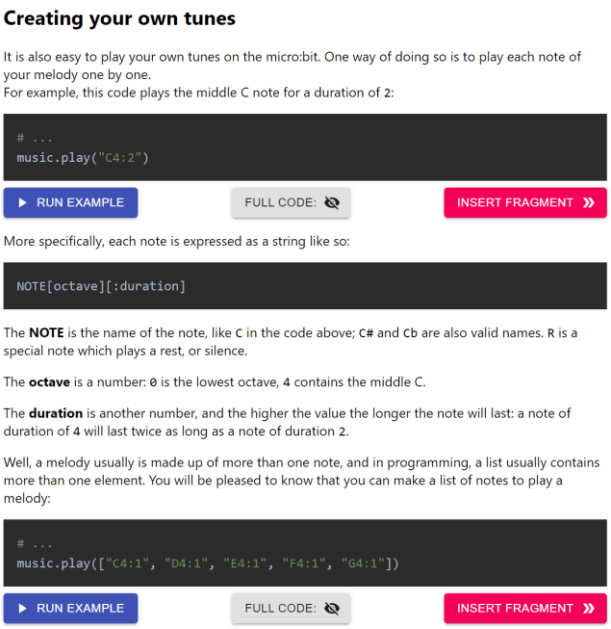
Firstly, there are multiple tutorials to choose from, and the user can switch between them using the SELECT TUTORIAL button. If they click it, they will see a menu pop-out on the left like this:



Here, the user can change tutorials by clicking one of the four listed below the 'Available Tutorials' title. For example, here is what the 'Python Language Features' tutorial looks like:



On a tutorial, there is a mixture of text and code:



Each snippet of code is accompanied by the RUN EXAMPLE and INSERT FRAGMENT buttons.

The RUN EXAMPLE button will run the code from the snippet on the Micro:bit. The INSERT FRAGMENT button will insert that bit of code into the editor (to the right). The top and bottom of some code snippets have commented-out ellipses:

This is to hide unnecessary parts of the snippet for the purposes of the tutorial. The user can reveal the hidden parts by clicking the FULL CODE button, for instance:

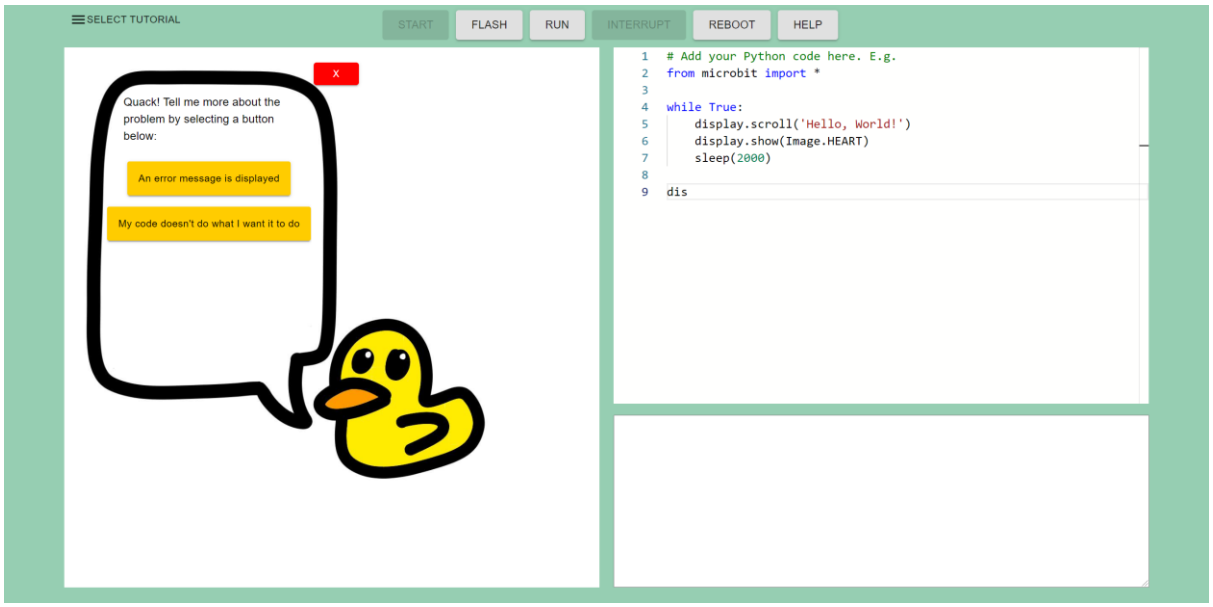
```
1  # LINES 4-6
2  import music
3
4  tune = ["G4:2", "G4:2", "A4:2", "Bb4:2", "G4:2", "R:2",
5          "F4:2", "F5:2", "R:2", "F5:2", "C5:2"]
6  music.play(tune)
```

RUN EXAMPLE FULL CODE: INSERT FRAGMENT

Using all this, they can go through the tutorial, reading and trying the code using a mixture of running snippets and running in the editor to what happens if they change things.

The Duck

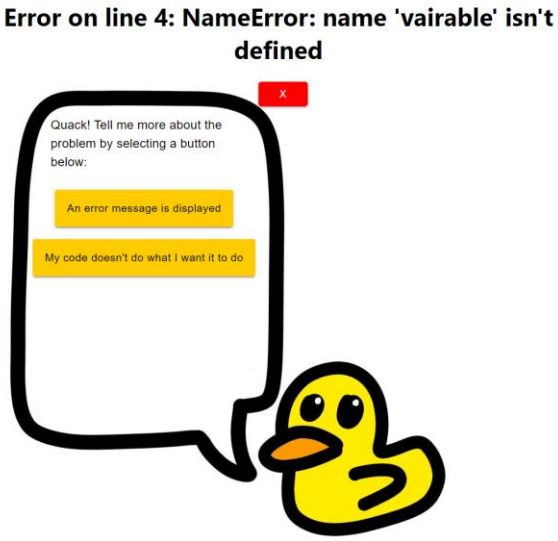
If the user clicks the HELP button, the helpful duck will pop up!



The duck will also come up if the user's code throws an error. For example:

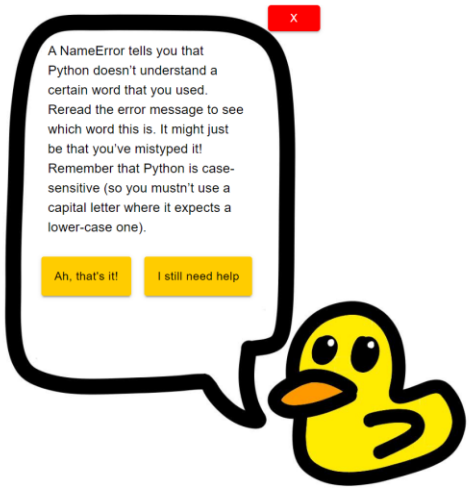
She guides the user through to a solution to their problem by asking questions to narrow down exactly what they need help with.

Let us run through an example, using the bug above. With that problem, the user might click ‘An error message is displayed’. Then they will see the following:



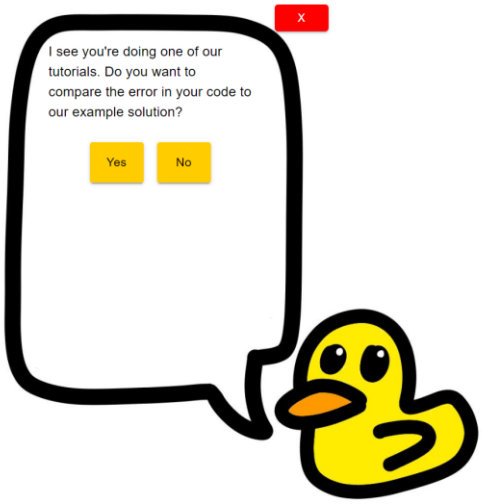
Then they will click ‘NameError’. The duck will be displaying this:

Error on line 4: NameError: name 'vairable' isn't defined



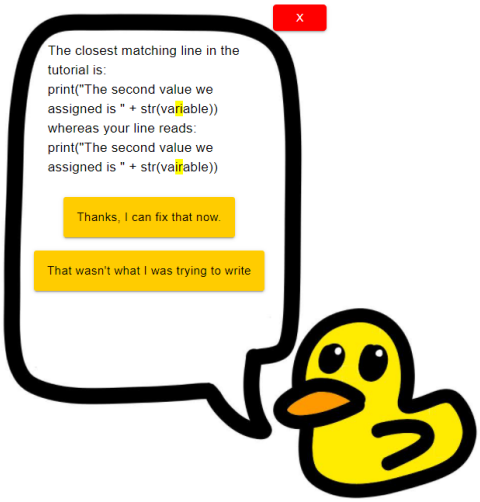
Now her message should help solve the user's problem! If not, and they still cannot debug their code, the user can click 'I still need help'. Then, she will ask them this:

Error on line 4: NameError: name 'vairable' isn't defined



If the user clicks 'yes' (assuming there actually is an error thrown from running the user's code), she will compare the erroneous line from the user's code to our 'example solution' for the current tutorial. From this, the user will get a comparison between their erroneous line and the closest matching line in the example solution:

Error on line 4: NameError: name 'vairable' isn't defined



Note the yellow highlighting, which should tell the user exactly what they need to add/change/remove to fix their line.

How to access & Delivery

A fully working implementation of the application is accessible via https://Microbit-team-12.github.io/editor/. Source code can be found on GitHub, at https://github.com/Microbit-Team-12/editor .
The delivery is automated; whenever a repository contributor pushes changes to the Master branch, GitHub Actions automatically builds the webpage and publishes to the webpage.

Support details

Third Party Libraries

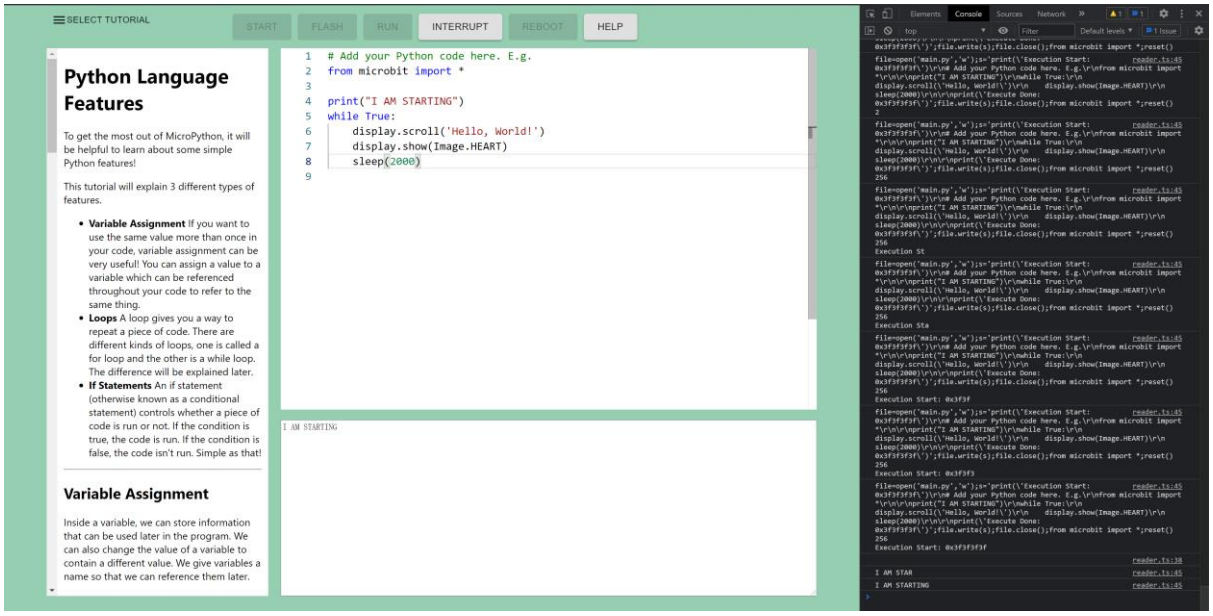
Library	Used For	License
fuse.js	Fuzzy string matching for when the Duck finds the closest matching line in a tutorial	MIT
material-ui	Visual appearance of buttons in user interface	MIT
monaco-editor	Basis of the text editor field on the right of the application	MIT
react	Creating the UI	MIT
react-markdown	Rendering the tutorials written in markdown	MIT
react-spaces	Layout of components inside speech bubble for Duck	MIT
react-syntax-highlighter	Syntax highlighting for embedded code inside tutorials	MIT
ts-stream	Used by the serial-based Micro:Bit flasher to pipe output from WebSerial	MIT
typescript	Writing better JavaScript code	Apache 2.0
types/jest **types/node** **types/react** **types/react-dom** **types/react-syntax-highlighter,** **types/w3c-web-serial**	Giving developers TypeScript function hints when developing the application	MIT
web-serial-polyfill	Providing webSerial over webUSB for browsers supporting webUSB but not webSerial	Apache 2.0

Troubleshooting Guide

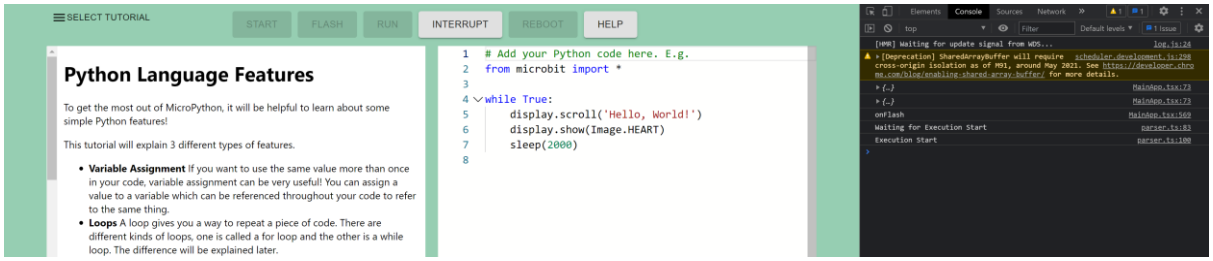
The serial-based Micro:bit flasher

Debugging:

When debugging the library, it may be helpful to see recent Micro:bit output. To enable this, the user needs to set `ManagerOption.readOption.showLog` to true.



You may also set `ManagerOption.signalOption.showLog` to true for state tracking log, although each log roughly starts at the beginning or end of the async function.



Common errors thrown by library:

The library is programmed in a defensive programming way. On wrong environment or state, errors are thrown by the library.

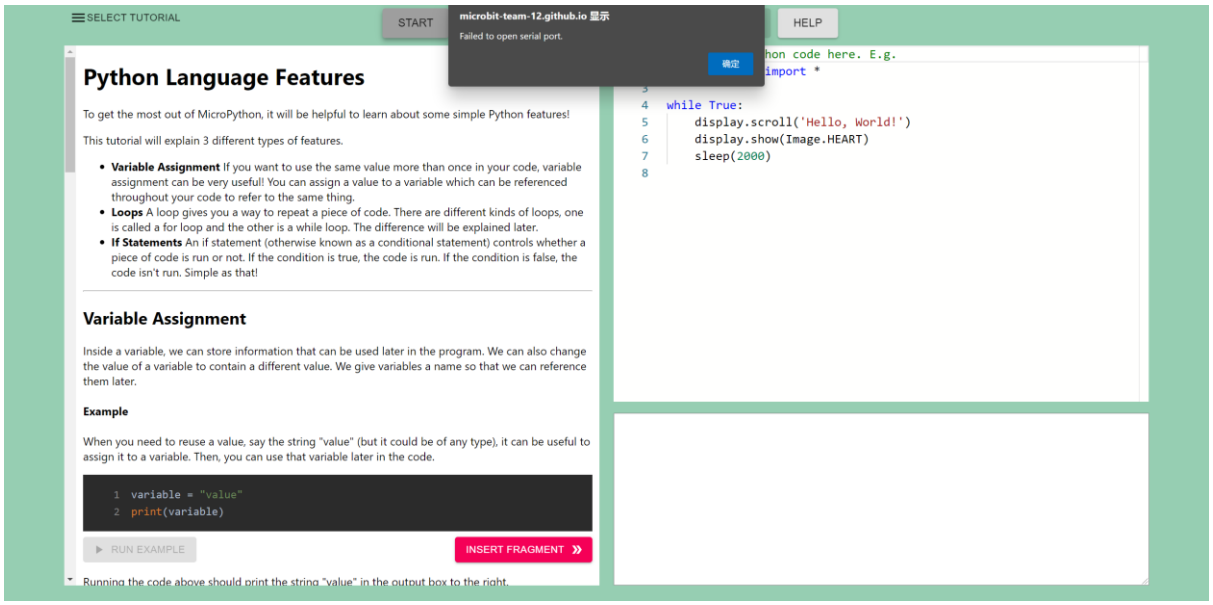
> "ConnectionFailure - Webserial is not supported": This is because current browser does not support WebSerial. The user should be informed to switch to a higher version of Chrome.
>
> "ConnectionFailure - Failed to Open Port": This is because the browser failed to obtain connection to the port. Often this is because some other programs (or webpages) occupy the connection.
>
> "ConnectionFailure - No Response on Validation": This is because the device the user connected does not certify this is a Micro:bit device, or the serial port is occupied and unavailable to the browser.
>
> "ExecuteFailed - Device not free": The intention of this error is to prevent multiple simultaneous executions of user code, which is not possible on the Micro:bit. Developers should enforce this on UI side, or interrupt before flash.

"SerialDisconnected": This is because the browser lostconnection to the device. It might be that the user unplugged the Micro:bit device.

On web

Error handling:

The UI implementation is supposed to prevent an ExecuteFailed error from occurring. On the other hand, connection failures cannot be resolved automatically and require user actions to resolve. In such cases, the browser alert window is used to display errors.



Known issues:

- WebSerial is recent technology, and it will take some time before every browser adopts it. Google's polyfill library does not work great either.
- After many rounds of interactions, the serial port may become unresponsive. Reconnecting the Micro:bit will resolve the problem.
- FOR DEVELOPERS: It seems that WebSerial is unresponsive in the first few seconds after Chrome is launched. This is what will happen when you click START for initial connection before WebSerial becomes responsive:
    - Trying to connect to a paired device. But this takes a few seconds until Chrome is ready for WebSerial.
        - Failed because there is not a paired connection.
    - Fallback to connect via the WebSerial permission window.
        - Error with Must be handling a user gesture to show a permission request. Because the clicking event becomes expired.

  The solution is to simply do your initial connection after a few seconds after chrome is ready. This is unlikely to affect normal browser users.

Testing

We decided to use manual, scripted testing to check that the UI-heavy part of our application dependent on a web-serial connection works as intended. The advantages to this type of testing are that:

1. We can specifically test different functionalities of our tool without simulating the web-serial connection in an automated test environment, would be very elaborate and time-consuming.
2. Our tool can be tested on the target audience and the customer which will provide us with rich feedback about the user experience.

A potential disadvantage is related to the lack of automated tests. Ideally, the testing would have been designed to perform every time a change was made to the application. However, our supervisor advised that it was out of the scope of this project to set up automated tests for the UI section of our project.

Below, the test scenarios have been outlined, with results from those who we were able to test it on ('Developer' is one of the team, 'Target Audience' is an 11-year-old student who carried out a subset of the tests).

Functional Area	**Test Name**	**Test Steps**	**Expected Results**	**Actual Results**
Set up of Micro:bit	Set up success	1. Connect Micro:bit using USB. 2. Press the START button to connect Micro:bit via web serial.	All buttons are now clickable except for START and INTERRUPT	*Developer:* expected *Target Audience*: expected
	Set up with no Micro:bit connected	1. Press the START button without connecting Micro:bit using USB.	A pop-up should tell you 'No Paired Serial Devices Available'	*Developer:* expected
Unclickable buttons	With code running	1. Connect the Micro:bit using the START button. 2. Press RUN at the top of the screen. 3. Click on one of START, FLASH, RUN, or REBOOT. 4. Click on HELP. 5. Click on INTERRUPT.	After step 3, nothing should happen. After step 4, the duck should appear. After step 5, the code should be interrupted and all buttons except START and INTERRUPT are clickable.	*Developer:* expected *Target Audience*: expected
Flashing code to Micro:bit	With an error	1. Connect the Micro:bit using the START button. 2. Type code in the text editor that contains an error. 3. Press the FLASH button.	The code flashes to the connected Micro:bit, but it will only display the error. The helpful duck should appear to give advice on how to deal with the error.	*Developer:* expected *Target Audience*: expected
	Without an error	1. Connect the Micro:bit using the START button.	The code should run on the Micro:bit and work as expected.	*Developer:* expected

		2. Type code in the text editor that does not contain an error. 3. Press the FLASH button.		*Target Audience*: expected
Editing code	While running on the Micro:bit	1. Connect the Micro:bit using the START button. 2. Clear the editor of all code except `from microbit import *`. 3. Press INSERT FRAGMENT underneath the first 'While loop' example in the Python Language Features tutorial. 4. Press RUN at the top of the screen. 5. Try to edit the code in the editor.	The code should not be able to be edited while running on the Micro:bit	*Developer:* expected *Target Audience*: expected
	Before running for the first time	1. Connect the Micro:bit using the START button. 2. Attempt to change the code in the editor.	The code should be able to be changed before running.	*Developer:* expected
	After running to completion	1. Connect the Micro:bit using the START button. 2. Clear the editor of all code except `from microbit import *`. 3. Press INSERT FRAGMENT underneath the second 'While loop' example in the Python Languages tutorial. 4. Press RUN at the top of the screen. 5. After the code has been run to completion, try to edit the code in the editor.	The code should be able to be changed after being run to completion.	*Developer:* expected
	After running and being interrupted	1. Connect the Micro:bit using the START button. 2. Clear the editor of all code except `from microbit import *`. 3. Press INSERT FRAGMENT underneath	The code should be able to be changed after running and being interrupted.	*Developer:* expected

		the first 'While loop' example in the Python Languages tutorial. 4. Press RUN at the top of the screen. 5. Press INTERRUPT 6. Try to edit the code in the editor.		
	After running and terminating with a SyntaxError	1. Connect the Micro:bit using the START button. 2. Clear the editor of all code except `from microbit import *`. 3. Press INSERT FRAGMENT underneath the second 'While loop' example in the Python Languages tutorial. 4. Change the final line to `print(x` rather than `print(x)`. 5. Press RUN at the top of the screen. 6. Try to edit the code in the editor.	After step 5, you should see the duck and a SyntaxError message. At step 6, the code should be able to be changed.	*Developer:* expected
Autocomplete	After typing 'a'	1. Connect the Micro:bit using the START button. 2. Type 'a' into the editor 3. Press tab	After step 2, an autocomplete box should pop up containing `accelerometer` and `audio`. After step 3, `accelerometer` should appear in your editor.	*Developer:* expected
	After typing 'T'	1. Connect the Micro:bit using the START button. 2. Type 'T' into the editor 3. Press tab	After step 2, an autocomplete box should pop up containing `True`. After step 3, `True` should appear in your editor.	*Developer:* expected
The helpful duck appears	Call on duck without an error	1. Connect the Micro:bit using the START button. 2. Press the HELP button. 3. Press 'My code doesn't do what I want it to do'. 4. Run through a path.	The helpful duck should appear to give advice with whatever is needed.	*Developer:* expected *Target Audience*: expected

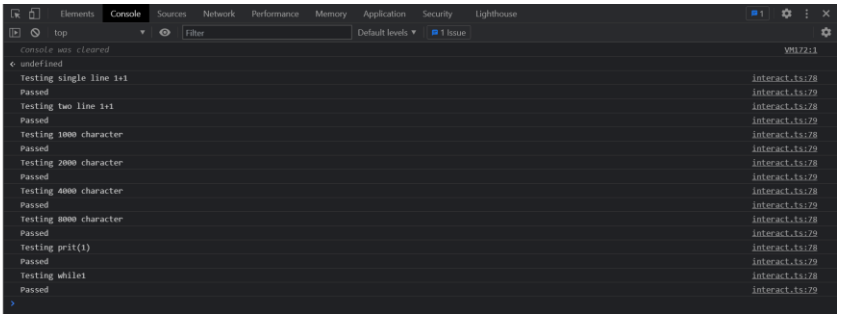
	Call on duck with NameError, without following a tutorial	1. Connect the Micro:bit using the START button. 2. Change line 5 to say `Display.scroll('Hello, World!')` 3. Press the RUN button.	After step 3, the duck should appear with the error message: `Error on line 5: NameError: name 'Display' isn't defined`.	*Developer:* expected
	Call on duck with a SyntaxError, without following a tutorial	1. Connect the Micro:bit using the START button. 2. Change line 5 to say `display.scroll('Hello, World!'` 3. Press the RUN button.	After step 3, the duck should appear with the error message: `Error on line 5: SyntaxError: invalid syntax`.	*Developer:* expected
	Call on duck with an ImportError, while following a tutorial	1. Connect the Micro:bit using the START button. 2. Clear the editor of all code except `from microbit import *`. 3. Press INSERT FRAGMENT underneath a piece of code in the Python Language Features tutorial. 4. Edit the code to make it have an error by replacing the word `microbit` with `Microbit`. 5. Press the RUN button. 6. Navigate through the duck, first pressing 'An error message is displayed', telling it nothing is helping until it compares your error with the tutorial.	After step 3, the code should appear in the editor. After step 5, the duck should appear, the error should be displayed, and the line with the error should be highlighted. After step 6, the difference between your code and the tutorial code should be highlighted within the duck's speech bubble.	*Developer:* expected *Target Audience*: expected
	Call on duck with a NameError, while following a tutorial	1. Connect the Micro:bit using the START button. 2. Clear the editor of all code except `from microbit import *`. 3. Press INSERT FRAGMENT underneath the first 'While loop' example in the Python	After step 3, the code should appear in the editor. After step 5, the duck should appear, the error should be displayed, and the line with the error should be highlighted. After step 6, the difference between your	*Developer:* expected

		Language Features tutorial. 4. Edit the code to make it have an error by replacing the function `display` with `displa`. 5. Press the RUN button. 6. Navigate through the duck, first pressing 'An error message is displayed', telling it nothing is helping until it compares your error with the tutorial.	code and the tutorial code should be highlighted within the duck's speech bubble.	
	Call on duck with a TypeError, while following a tutorial	1. Connect the Micro:bit using the START button. 2. Clear the editor of all code except `from microbit import *`. 3. Press INSERT FRAGMENT underneath the first 'While loop' example in the Python Language Features tutorial. 4. Edit the code to make it have an error by replacing the integer `500` with the string `'500'`. 5. Press the RUN button. 6. Navigate through the duck, telling it nothing is helping until it compares your error with the tutorial.	After step 3, the code should appear in the editor. After step 5, the duck should appear, the error should be displayed, and the line with the error should be highlighted. After step 6, the difference between your code and the tutorial code should be highlighted within the duck's speech bubble.	*Developer:* expected
The helpful duck disappears	Make the duck disappear by pressing the red 'X'	1. Press the HELP button. 2. Press the red 'X'.	After step 2, the duck should disappear.	*Developer:* expected *Target Audience*: expected
	Make the duck disappear by pressing 'Goodbye' at	1. Press the HELP button. 2. Press 'My code doesn't do what I want it to do'.	After step 5, the duck should disappear.	*Developer:* expected

	the end of the flowchart	3. Press 'A list has changed even though I didn't change it'. 4. Press 'Ah, that's it!' 5. Press 'Thanks Duck, bye for now!'		
	After making the duck disappear by pressing the red 'X' halfway through a path, make the duck reappear	1. Press the HELP button. 2. Press 'My code doesn't do what I want it to do'. 3. Press 'A list has changed even though I didn't change it'. 4. Press the red 'X'. 5. Press HELP again.	After step 5, the duck should start back from the beginning of the flowchart. Rather than the slide it was closed from.	*Developer:* expected *Target Audience*: expected
Testing code from tutorial	Running code directly from tutorial	1. Connect the Micro:bit using the START button. 2. Press a RUN EXAMPLE from within the tutorial.	After step 2, the example should run on the Micro:bit and behave as expected. The code isn't expected to stay on the Micro:bit when disconnected from the computer	*Developer:* expected *Target Audience*: expected
	Inserting a fragment, then flashing	1. Connect the Micro:bit using the START button. 2. Clear the editor of any code except `from microbit import *`. 3. Press INSERT FRAGMENT underneath a piece of code in the Python Language Features tutorial. 4. Press the FLASH button. 5. Disconnect the Micro:bit.	After step 3, the code fragment should appear in the editor. After step 4, the code should be flashed to the Micro:bit, behave as expected. After step 5, the program should continue to run on the Micro:bit once reconnected.	*Developer:* expected

A separate webpage (TestPage.tsx) was created to run code tests on the serial-based Micro:bit flasher. When the Micro:bit is connected to the computer via USB, a library developer can press a button on this page to run the test. This test flashes a few snippets of python code to the device and checks if the serial output matches what is expected. This covers a wide range of paths, including interrupting a 'while True' loop, exceptionally large files, print statements, etc. The result of these tests was printed to the console as shown below.

START



Is it adaptable for the future?

We used JSDoc for our function header comments, so that future developers can understand what each function does. This means that our code is comprehensible and easily extendible in the future. Furthermore, we separated different modules into different files, so that each one can be lifted independently in the future.

The serial-based Micro:bit flasher is a module with minimal dependency that can also be used in other Micro:bit editor projects, separate from our application.