adafruit learning system

Adafruit WebIDE

Created by Tyler Cooper

Last updated on 2018-08-22 03:31:53 PM UTC

Guide Contents

Guide Contents 2
Overview 3
Installation 4
- Easy installation: 4
- Manual Installation: 4
- Uninstallation: 4

Getting Started 6
Using the WebIDE 7
Using the Debugger 10
Using the Visualizer 12
- Starting the visualizer 12

FAQ 14
- Does the webIDE support python3? 14
- How do I change the http port the WebIDE is using? 14
- How can I fix my repository when it fails to commit or push to a remote repository? 14

Submit a Bug 15

Overview

We finally have the Adafruit WebIDE in a place where we think it's solid enough for a beta release. First, a disclaimer: this release will likely have bugs, and minor issues.

The Adafruit WebIDE is by far the easiest way to run code on your Raspberry Pi or BeagleBone. Just connect your Pi or BeagleBone to your local network, and log on to the WebIDE in your web browser to edit Python, Ruby, JavaScript, or anything and easily send it over to your Pi. The WebiDE includes a terminal, so you can easily send various commands to your Pi right from the browser. Also, your code will be versioned in a local git repository, and pushed remotely out to any remote git repository so you can access it from anywhere, and any time.

Installation

Installation of the editor can be performed in two ways. One is the more trusting, but much easier way, the second is a bit more manual.

Easy installation:

Log into your Raspberry Pi. If you're on a Mac, you can open Terminal.app to log into the Raspberry Pi over SSH. Linux users can open the default terminal application. If you're using Windows, you'll want to download a good terminal application. My favorite is PuTTY (https://adafru.it/aWh).

Once you have the terminal application open, assuming you're using Occidentalis, type in the following:

```
$ ssh pi@raspberrypi.local
pi@raspberrypi.locals password:
```

Once you type your password in, and get a prompt, you can copy and paste the following command, and hit enter. This command will download an install.sh script from our github repository, and execute it automatically for you.

```
    curl https://raw.githubusercontent.com/adafruit/Adafruit-WebIDE/master/scripts/install.sh | sudo sh
```

The editor will be installed into /usr/share/adafruit/webide using the user webide. The script will install node, npm, redis-server, git, and i2c-tools. If you'd like to review the script, it's located in our repository (https://adafru.it/aQo).

The installation may take from 3-5 minutes, so please be patient.

After the installation is complete, you'll see the following prompt:

```
**** Starting the server...(please wait) ****
**** The Adafruit WebIDE is installed and running! ****
**** Commands: sudo systemctl {start,stop,restart} adafruit-webide  ****
**** Navigate to http://raspberrypi.local:8080 to use the WebIDE
```

The editor is now installed, and you can open a browser to access it from any computer in your network.

Due to our very small development team, and limited resources, the only browsers that are supported are Google Chrome, and Mozilla Firefox at this time. We hope to support more in the future!

Manual Installation:

You can manually install the editor by following along in the following installer script and choosing the components you'd like to install:

```
https://raw.githubusercontent.com/adafruit/Adafruit-WebIDE/master/scripts/install.sh
```

Uninstallation:

To uninstall the editor you can run the following script:

```
curl https://raw.githubusercontent.com/adafruit/Adafruit-WebIDE/master/scripts/uninstall.sh | sh
```

You can also manually uninstall by removing the following components:

- Delete the folder the editor exists in.
- Uninstall nodejs npm redis-server git avahi-daemon i2c-tools.

Getting Started

> The WebIDE is still in beta, and not designed to be used outside of your private secure network or exposed to the internet.

Setting up your Rapsberry Pi WebIDE will only take a few minutes. Let's get started.

First up, you will connect to the WebIDE (running on the Pi) using a **different computer** (not the Pi). The client computer must be on the same network.

Open up a web-browser on a computer that shares the same networks as the Pi and browse to either **http://raspberrypi.l (https://adafru.it/aWs)ocal (https://adafru.it/CbY)**:8080 (https://adafru.it/CbY) if you are running Raspian/Debian Stretch, or to **http://xx.xx.xx.xx** (https://adafru.it/c59) where the x's are replaced with your Pi's IP address.

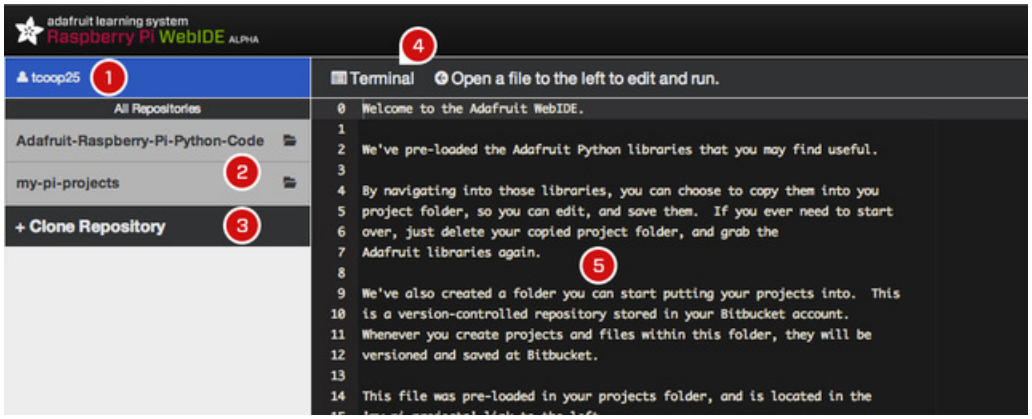
Using the WebIDE

> Once you have completely installed the WebIDE, type http://raspberrypi.local:8080 or http://beablebone.local:8080 into your browser.

The Adafruit Learning System Raspberry Pi WebIDE is packed with neat features. Many of which aren't all that obvious. Here is a running list of all of those features, and how they work.

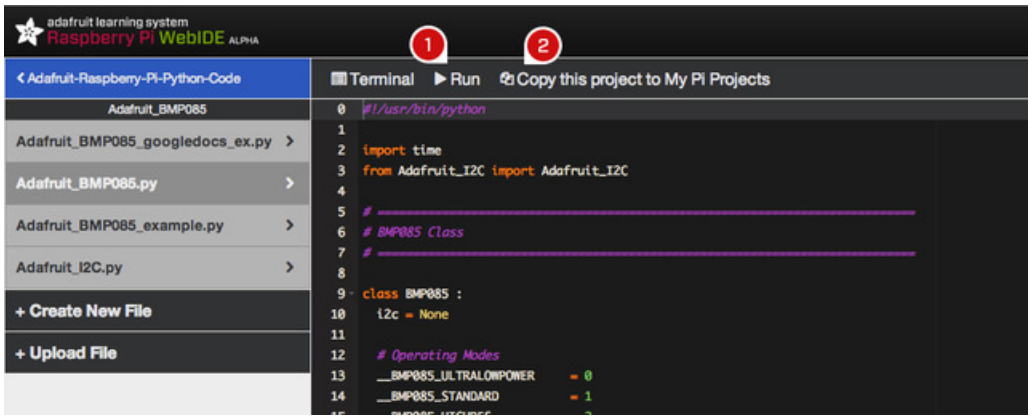
> As the WebIDE is still in its infancy, we need your help to find and squash bugs. Use the link below to submit an issue on Github.

https://github.com/adafruit/Adafruit-WebIDE/issues (https://adafru.it/aQ8)



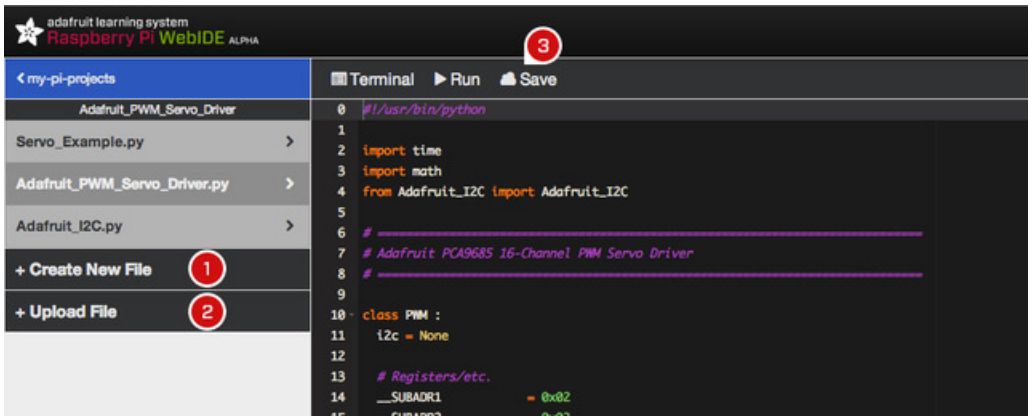
When you first load up the WebIDE for the first time, you should see something similar to what you see above (click on the image to view it in its full size).

1. Here is where you can view your editor settings.
2. Here is a list of all of your repositories. The Adafruit Raspberry Pi Python Code repository (https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code.git (https://adafru.it/aYZ)) is automatically cloned into your Bitbucket account. We have also created a my-pi-projects repository. If it has the little folder icon, that means there are files contained within. If it has an arrow icon, that means it is a file that can be viewed in the WebIDE.
3. Click here to clone a repository from Github or Bitbucket. Follow the instructions in the popup.
4. No matter where you are in the WebIDE, there will always be a terminal button. Click this to talk directly to your Raspberry Pi, or do things like install libraries without leaving the WebIDE. Handy!
5. When you first log in (or refresh your WebIDE in your browser), you will be greeted with a nice message. Read this for any news on the WebIDE.



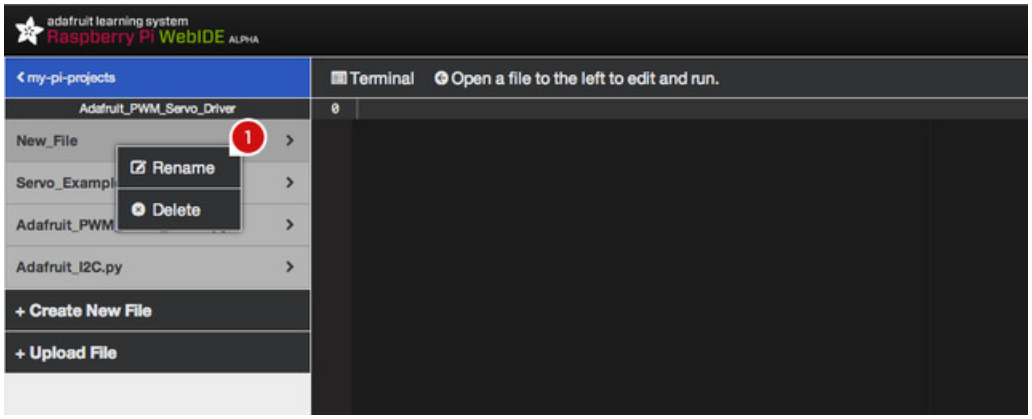
When you view any of the included code examples from Adafruit, there are a couple unique items.

1. As with any piece of code, you can simply click the Run button to execute the code on your Raspberry Pi. The terminal will automatically open when you click Run.
2. The Adafruit code is read only, but you can easily copy this code to your my-pi-projects repository. Just click this link and it will automatically copy the folder, and all of its contents over.



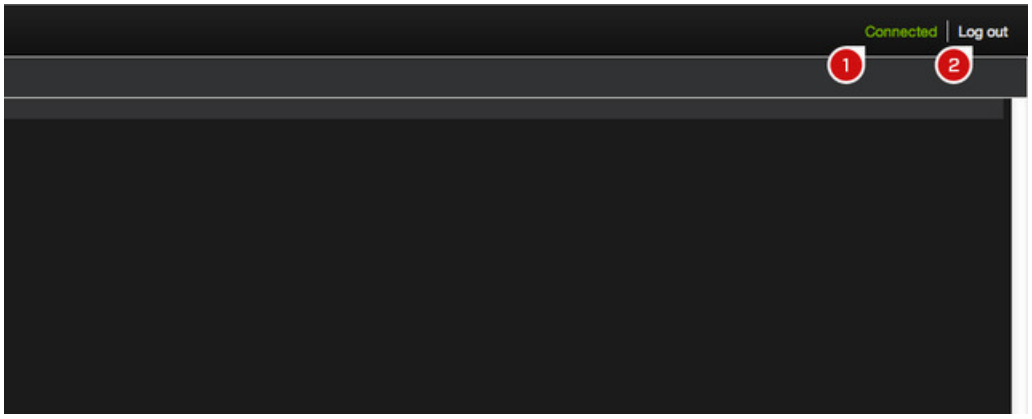
Once you have copied, imported, or created code from scratch in your my-pi-projects (or other repository), this is the screen you will see.

1. If you want to create a new file, press this button, and name your file (make sure to add a proper file type, like .py to the end of your name, otherwise the editor won't know what to do with it). Then click save to add your new file to the current folder.
2. You can also upload a premade file. Once again, make sure it is named properly before importing. You can also import images to the folder (and view those images right in the IDE).
3. Make sure to click save after making any changes to code. This will then update the files in your Bitbucket account.



Renaming and deleting files is easy!

1. Just right click on any of the files that you want to change and the following box will pop up. Click delete to permanently delete that file. If you click rename, you will be asked what name you want the file to change to. Make the change, then click save.



In the upper right of the WebIDE, you will see this.

1. If it is green, and says 'Connected', that means the WebIDE is communicating with your Raspberry Pi. If it is disconnected, the WebIDE will keep trying to make contact with your Pi for awhile, then it will finally give up. It will then ask you to reconnect your Pi, and refresh the browser.
2. You can log out of your Bitbucket account at any time by clicking Log out.

What isn't shown in the image above is our auto-update feature. If we push out a new update for the WebIDE, you will see it pop up between 'Connected' and 'Log out'. Simply click the link, sit back, and the WebIDE will automatically download, unpack, install, restart your Raspberry Pi, and even refresh your browser for you. It doesn't get any easier than that.

In order to see if there are updates available, refresh your browser every day.

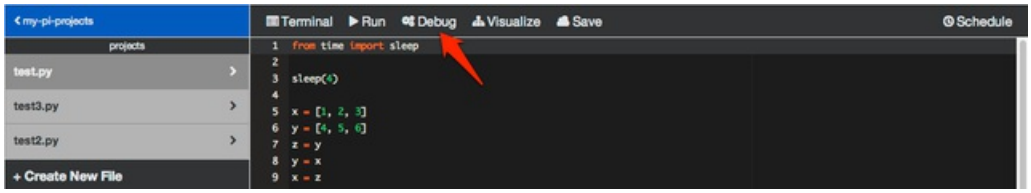
Once the WebIDE is installed, you can shutdown (shutdown -h now) or restart (shutdown -r now) your Pi anytime. The next time you want to use it, just start your Pi, and reload **raspberrypi.local** to log back into the WebIDE.

Using the Debugger

The python debugger in the WebIDE can be quite useful for many situations. The debugger allows you to step through your python program in realtime. This is in contrast to the visualizer, which runs the program fully, and then allows you to step through it.

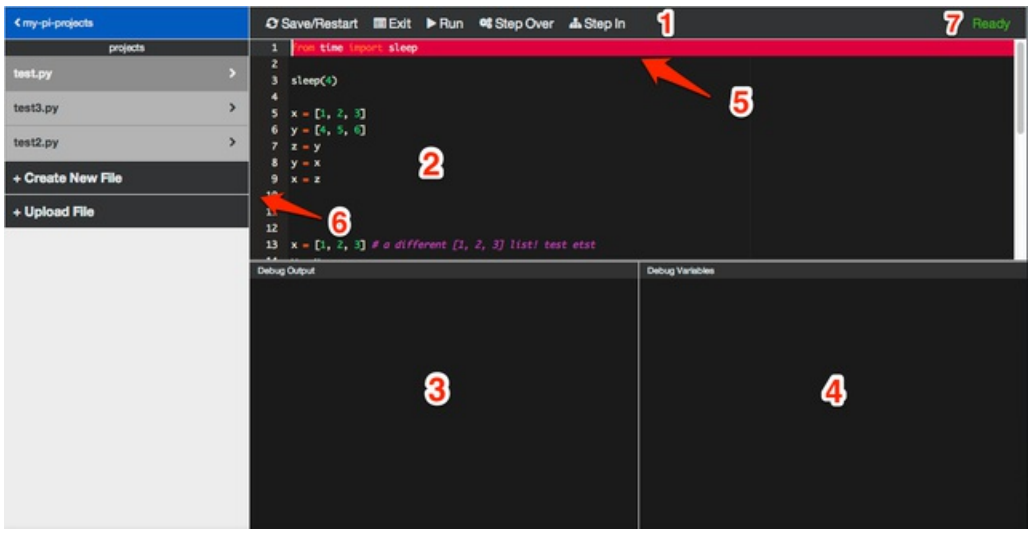
When would you use the debugger? One scenario would be if you're just not getting the right values returned out of your functions. Or your variables aren't being set to the values you'd expect. You could then debug it, set a breakpoint to where you think it's broken, and see exactly what the values are at that given moment in your program. Also, you could use the GPIO pins to light various LED's. As you step through your program, you'll see them turn on and off as you'd expect.

To use the debugger, open a python file, and click the "Debug" link in the toolbar:



Once you've clicked the link, the toolbar will change with a new debugging bar. It should open within a couple of seconds. The editor will still display, and you can make changes to your code while debugging.

Once the debugger is ready, you'll see the following:



There is quite a bit more to the debugger than just running a program, or even than using the visualizer.

The first think you'll notice is the toolbar (#1) has new buttons available:

1. Save/Restart allows you to save your file, and restart the debugger for that file in one click. You can initiate that by hitting control-s, or command-s (on OS X).
2. The Exit link will exit the debugger.
3. The Run link will continue execution of your program either to the end, or to the next breakpoint you have enabled. If there is a breakpoint enabled (clicking in the gutter (#6)), it will stop there, and wait for your instruction.
4. Step Over will step through your program line-by-line until it hits the end. This option will skip any function calls.

So, if you have a function called foo(), it will just run the entire function, and move to the next line.

5. Step In will step through your program as well, but it will jump into many of the functions, instead of skipping over them.

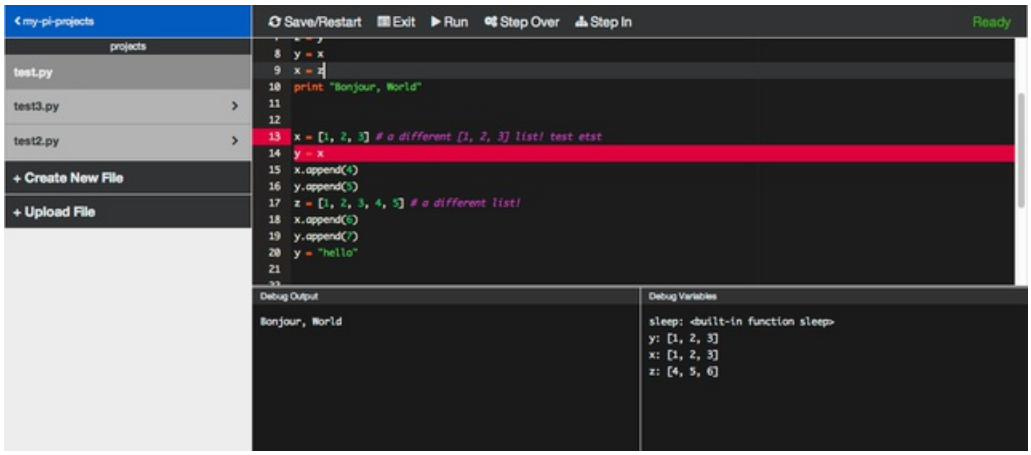
The editor is also displayed (#2). You can edit your file at any time. If you do change your file, you'll need to click the Save/Restart link in the toolbar in order to let the debugger pick up any of the changes you've made.

At the bottom two panels, you'll see the Debug Output (#3), and the Debug Variables (#4). The Debug Output displays anything from stdout or stderr that your program would output (such as print statements). The Debug Variables displays the live variables as they're being assigned.

The red line in the editor (#5) moves as you step through your program. It is the line that will execute next, and will stay centered in the editor window as you step through your program.

At the far left of the editor (#6), there is a blank space to the left of the line numbers. This is called the gutter. If you click in the gutter when the debugger is in a "Ready" state, you can add breakpoints. A breakpoint is useful for many situations, such as if you have a longer script, and there is a certain troubling section that is buggy. Instead of slowly stepping through your program you would set the breakpoint (a red square will appear in the gutter), and then click "Run". The script will execute to the breakpoint, and the red line will stop, and wait for you to continue stepping through it.

The last feature you'll want to be aware of is the debug status messages (#7). These will show you what the debugger is doing. For example, when it initially loads, or when you click "Save/Restart", it will be in an "Initializing..." state. When it's ready for your input, it will be "Ready". Some parts of your script will cause it to appear locked up, but it's really waiting for the server to return a response from your script (a long sleep() statement could cause this, for example).



In the screen shot above, you can see what the debugger looks like in the middle of running a program. The debugger is currently one step after the breakpoint that was set on line #13. You can see the 'print "Bonjour, World"' in the Debug Output, and the variables as they are in that given state of the program execution.

At this point in the debug cycle, you could edit your file, and Save/Restart. You could continue stepping through the program, or you could Exit the debugger.

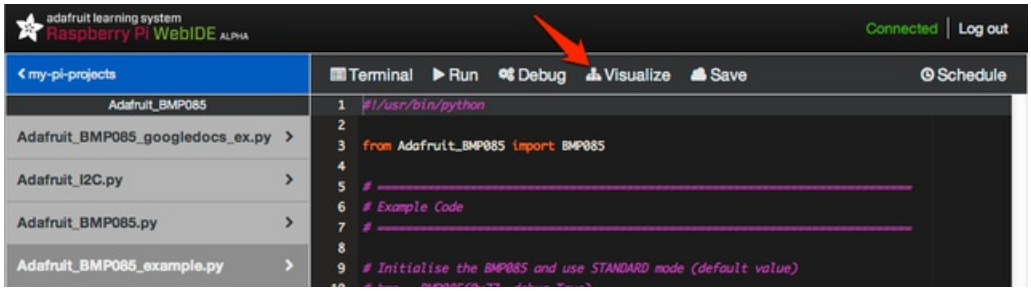
Using the Visualizer

The visualizer is a feature designed to help understand how a Python program is working at a more basic level. It let's you see what the python interpreter is doing as it steps through your program, such as variables being assigned, objects being created, etc.

Generally, the visualizer is a good tool to use if you have simpler scripts that you'd like to understand. Some of the questions it will help you with are: "How does recursion work?", "How does a function call work?", "How do various methods, such as append, work when working with lists?".

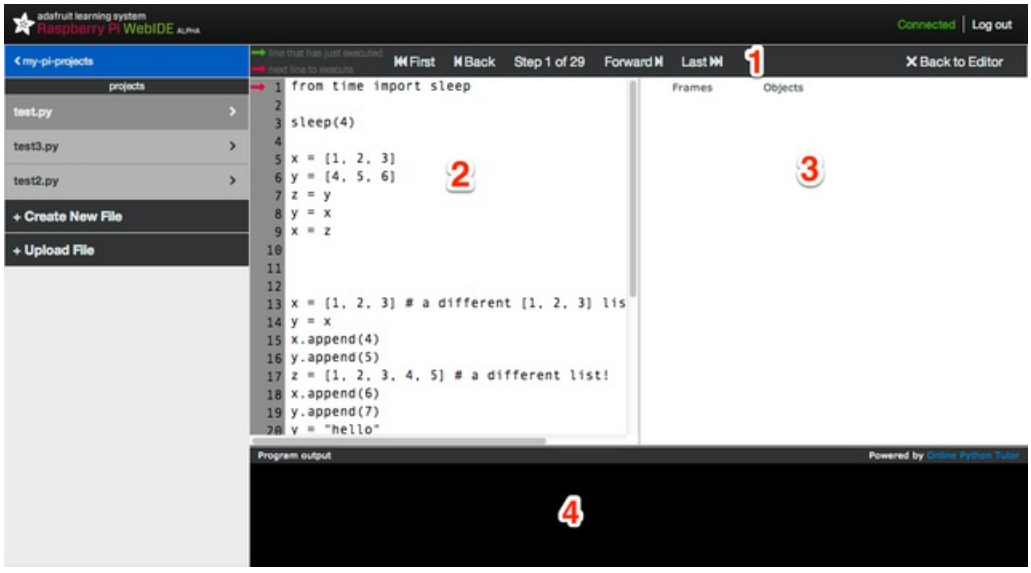
Starting the visualizer

In order to start the visualizer, open a python script that you're able to run. In the menu, there will be a link with the title "Visualize". Click the "Visualize" link.



Once you've clicked the link, it will take from a couple of seconds, to a few minutes to load, depending on what your python script is doing. Something simple, like lighting a few LED's, or printing out a few lines will load very quickly. One thing to watch out for is that you don't have any infinite loops in your code, such as if you have a script that is designed to just keep running while periodically checking the temperature.

The next screen you'll see is the following:



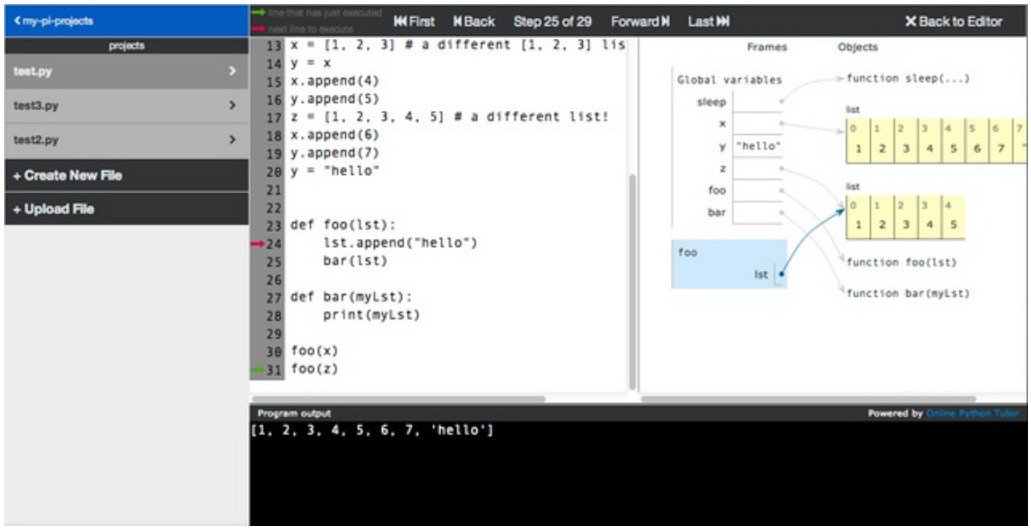
The toolbar (#1) allows you to step through your program. You can navigate any way you'd like, forward, back, and jump to the start, or end.

The code (#2) is listed to the left. As you step through your program, you'll see two arrows in the left gutter that will show you what line has just executed (green), and what will be executing once you click "Forward" (red).

The right column displays the stack (#3) as your program executes. You'll see your variables getting assigned, objects being created, etc.

The bottom pane is your program output (#4). Anytime you have a print statement, or any other type of output, you'll see it displayed in this section.

Here is a screen shot of how it looks while stepping through a program:



One thing to note about the visualizer is that it runs your entire script on the server, then sends the playback of your script to the WebIDE. It's not a real-time debugger (but the WebIDE has one of those as well!). For example, if you have an LED that lights up in your program, it will light up quick and then send back the response, and when you step through your program, the LED won't light up as it has already run.

Credit to the visualizer is given to Philip Guo at pythontutor.com (https://adafru.it/aVp). We forked, tweaked, converted, and modified his open source creation to work as a streamlined feature within the WebIDE.

FAQ

Does the webIDE support python3?

Not yet, but our goal is to eventually support it. It's mostly a matter of balancing the bug fixes with the feature requests. That being said, we do accept pull requests! :)

How do I change the http port the WebIDE is using?

The default is port 8080. You can change the port by editing the systemd service at the following location:

`/etc/systemd/system/adafruit-webide.service`

How can I fix my repository when it fails to commit or push to a remote repository?

This can happen any number of ways, such as with an SD card that is bad, or if you shut your system down while a commit or push is in progress, or any number of other ways.

You can find out what the real issue is by viewing the log file.

Open the terminal in the webide and execute the following: cd /usr/share/adafruit/webide/logs

Then execute the following command to view the last few commands from the log file: cat output.log

You can also view the errors.log file: cat errors.log

If you see an error such as:

```
error: object file .git/objects/2c/e98048b95181870dbba33675f3d2bc1d995c47 is empty
fatal: loose object 2ce9804 (stored in .git/objects/2c/e98048b95181870dbba33675f3d2bc1d995c47) is corrupt
```

You can try fixing that by following this fix from Stack Overflow.

Submit a Bug

Submit a Bug (https://adafru.it/aQ8)