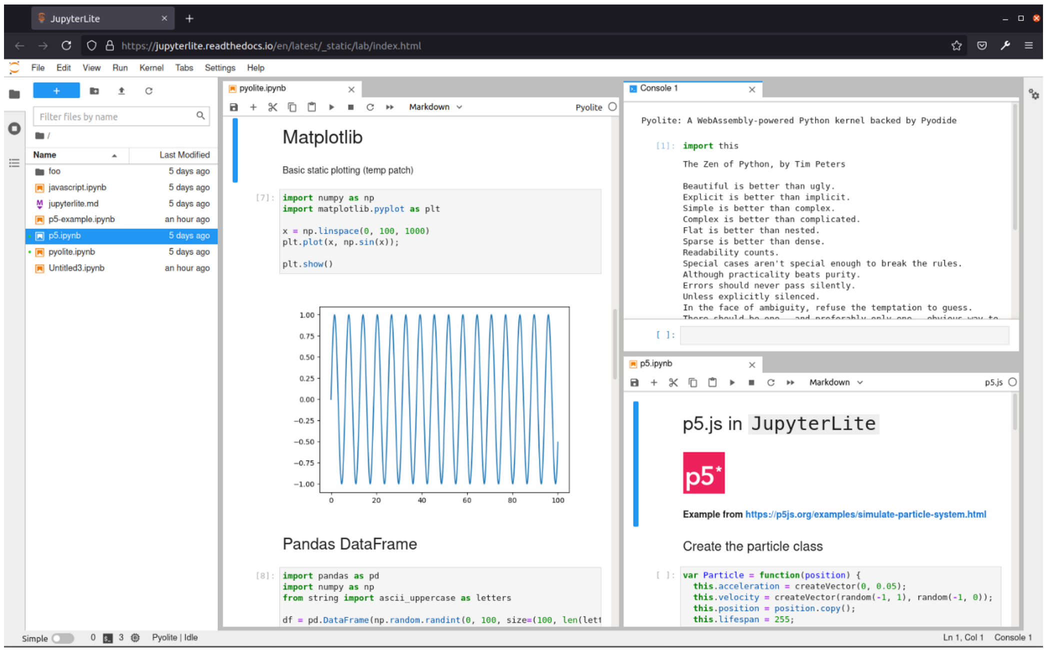
Published in Jupyter Blog

Thorsten Beier Follow
Jul 14 · 8 min read · Listen

Mamba meets JupyterLite

Introducing a mamba-based distribution for WebAssembly, and deploying scalable computing environments with JupyterLite.

JupyterLite is a Jupyter distribution that runs entirely in the web browser without any server components. To achieve this, all language kernels must also run in the browser.


JupyterLite
https://jupyterlite.readthedocs.io/en/latest/_static/lab/index.html
File Edit View Run Kernel Tabs Settings Help
Filter files by name
pyolite.ipynb
Markdown
Pyolite
Console 1
p5.ipynb
p5.js
Simple 0 3 Pyolite | Idle
Ln 1, Col 1 Console 1

Name	Last Modified
foo	5 days ago
javascript.ipynb	5 days ago
jupyterlite.md	5 days ago
p5-example.ipynb	an hour ago
p5.ipynb	5 days ago
pyolite.ipynb	5 days ago
Untitled3.ipynb	an hour ago

Matplotlib

Basic static plotting (temp patch)

```
[7]: import numpy as np
     import matplotlib.pyplot as plt

     x = np.linspace(0, 100, 1000)
     plt.plot(x, np.sin(x));

     plt.show()
```

Pandas DataFrame

```
[8]: import pandas as pd
     import numpy as np
     from string import ascii_uppercase as letters

     df = pd.DataFrame(np.random.randint(0, 100, size=(100, len(lett
```

```
Pyolite: A WebAssembly-powered Python kernel backed by Pyodide

[1]: import this

     The Zen of Python, by Tim Peters

     Beautiful is better than ugly.
     Explicit is better than implicit.
     Simple is better than complex.
     Complex is better than complicated.
     Flat is better than nested.
     Sparse is better than dense.
     Readability counts.
     Special cases aren't special enough to break the rules.
     Although practicality beats purity.
     Errors should never pass silently.
     Unless explicitly silenced.
     In the face of ambiguity, refuse the temptation to guess.

[ ]:
```

p5.js in `JupyterLite`

Example from https://p5js.org/examples/simulate-particle-system.html

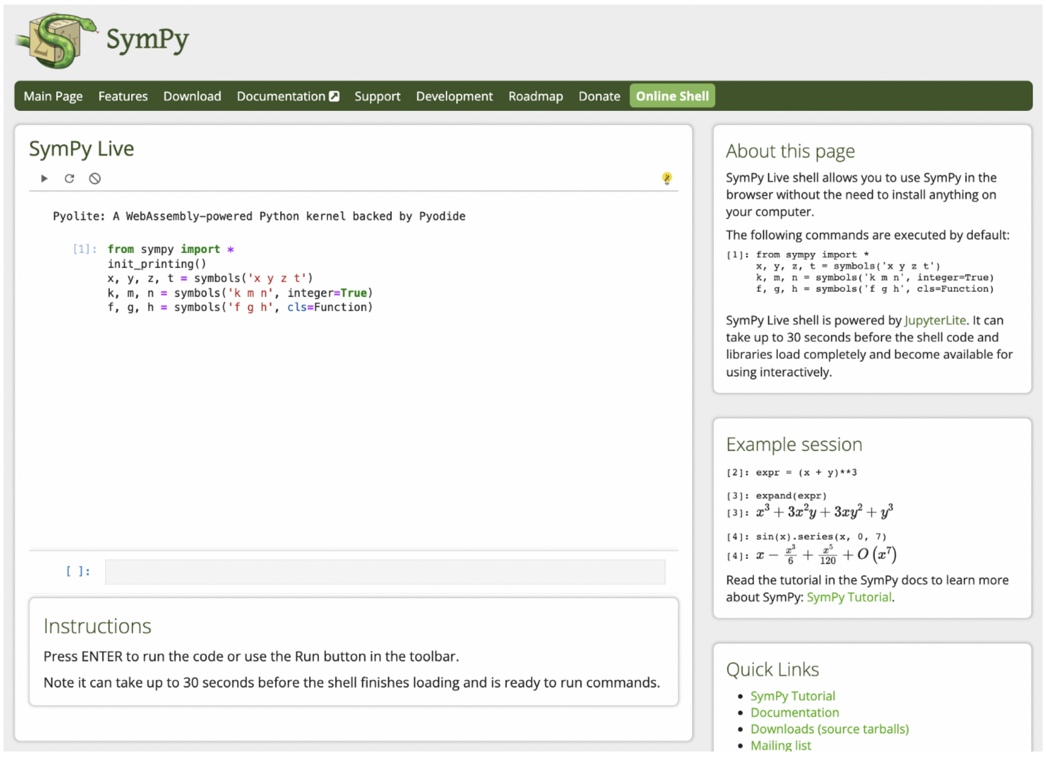
Create the particle class

```
[ ]: var Particle = function(position) {
       this.acceleration = createVector(0, 0.05);
       this.velocity = createVector(random(-1, 1), random(-1, 0));
       this.position = position.copy();
       this.lifespan = 255;
```

JupyterLite running in the browser as a static website collection of static assets. It makes it possible to embed a console or a notebook interface on any static page or blog without having to deal with a server architecture deployment. The **scalability** of this approach allowed several major projects of our ecosystem (NumPy, SymPy, Pandas, PyMC, and many more) to embed interactive examples on their websites, which are visited by millions of users monthly.

> JupyterLite is the easiest and most scalable way to embed an interactive console or notebook on a web page without any server component.

The most prominent JupyterLite kernel is the *Pyolite* Python kernel, which is based on the Pyodide distribution for WebAssembly. Beyond the CPython interpreter, Pyodide includes many popular scientific computing packages such as NumPy, Pandas, and Matplotlib. Pyodide also provides a foreign function interface (FFI) that allows calling Python from JavaScript and vice versa.



SymPy Live

```
Pyolite: A WebAssembly-powered Python kernel backed by Pyodide

[1]: from sympy import *
     init_printing()
     x, y, z, t = symbols('x y z t')
     k, m, n = symbols('k m n', integer=True)
     f, g, h = symbols('f g h', cls=Function)

[ ]:
```

Instructions

Press ENTER to run the code or use the Run button in the toolbar.

Note it can take up to 30 seconds before the shell finishes loading and is ready to run commands.

About this page

SymPy Live shell allows you to use SymPy in the browser without the need to install anything on your computer.

The following commands are executed by default:

```
[1]: from sympy import *
     x, y, z, t = symbols('x y z t')
     k, m, n = symbols('k m n', integer=True)
     f, g, h = symbols('f g h', cls=Function)
```

SymPy Live shell is powered by JupyterLite. It can take up to 30 seconds before the shell code and libraries load completely and become available for using interactively.

Example session

```
[2]: expr = (x + y)**3

[3]: expand(expr)
```

[3]: $x^3 + 3x^2y + 3xy^2 + y^3$

```
[4]: sin(x).series(x, 0, 7)
```

[4]: $x - \frac{x^3}{6} + \frac{x^5}{120} + O\left(x^7\right)$

Read the tutorial in the SymPy docs to learn more about SymPy: SymPy Tutorial.

Quick Links

- SymPy Tutorial
- Documentation
- Downloads (source tarballs)
- Mailing list

The JupyterLite inline console embedded on the SymPy project website of pure python packages installed on top can be set. Our goal is to enable the **composability of computing environments** allowed by package managers and to adopt the conda-forge model for large-scale software distribution crowdsourcing.

Being able to pin down package versions in an environment is a strong requirement for software **reproducibility.** In fact, a locked WebAssembly environment could be seen as a reproducibility **time capsule**. As WebAssembly is a recognized web standard, it ought to be runnable for much longer than native binary packages: these are bound to a combination of architecture and platform and will eventually require an emulator.

This is why we developed a mamba-based distribution of WebAssembly packages built with Emscripten.

Emscripten-forge

The choice of the Mamba/Conda package manager was natural. Its main strength is the conda-forge community-maintained distribution, which has become the *de facto* standard source of packages for scientific computing.

> *Beyond its solid technological foundations and the multi-platform nature of the conda-forge distribution, its main strength is its social model. It allowed for a crowdsourcing approach of the packaging problem, with a balance of separation of concerns between maintainer teams and across-the-board automation, plus an amazing maintainers community.*
>
> *We plan on contributing this work to the conda-forge project, so that all recipes live in the same space.*

The Mamba/Conda package manager has support for many platforms and architectures such as Linux, OS X (for both x86 and arm64), and Windows. However, the **WebAssembly** family of platforms is not supported yet.

Adding support for WebAssembly to mamba & conda

To create conda packages for the WebAssembly platform, we relied on the Emscripten toolchain.

We then associated the Emscripten compiler, wrapped in a conda package as the C/C++ compiler for this new target.
This already allowed us to build many packages, including simple libraries like `bzip2` and `zlib`, but also more complex packages like `Python`.

- For Python extension modules we used `crossenv` which can create virtual environments for cross-compiling, and `cross-python` which integrates `crossenv` into conda. All the code and recipes for cross-compilation are hosted on the emscripten-forge GitHub repository.
- We then used GitHub actions to build packages with Emscripten and upload them to a package server.
- Packages are hosted on a deployment of the Quetz open-source server.

With this, you can easily create an environment for the `emscripten-32` target:

```
micromamba create -n my-env --platform=emscripten-32 \
    -c https://repo.mamba.pm/emscripten-forge \
    -c https://repo.mamba.pm/conda-forge \
    python ipython numpy jedi
```

Note that we not only added emscripten-forge as a channel, but also conda-forge. This means all noarch packages can be used.

Adding new packages to the emscripten-forge channel

Adding new packages is a simple procedure:

- fork the repository https://github.com/emscripten-forge/recipes
- create a folder for your package in **recipes/recipes_emscripten/<my_package>**
- add a `recipe.yaml` for your package in `recipes/recipes_emscripten/<your_package>`
- create a pull request containing the recipe. Once the pull request is merged, the package is automatically uploaded to the `emscripten-forge` channel.

Integration with JupyterLite

Even though this is a general-purpose conda-based distribution for Emscripten

The main reason for picking xeus-python (over ipykernel) is that with xeus-based kernels, it is possible to override the communication layer of the kernel (switching e.g. from ZMQ to HTTP/2). In the case of JupyterLite, the implementation simply relies on direct JavaScript function calls.

You can check out our earlier blog post for more details on the JupyterLite xeus-based kernels.

Providing a complete Python development experience

Some remaining intrinsic limitations to the WebAssembly platform need to be worked around to provide a complete experience to end-users. For example, sockets cannot be created in WebAssembly, preventing the use of the default asyncio event loop implementation. Luckily, the Pyodide authors developed a custom asyncio event-loop called WebLoop: it wraps the browser event loop using the Python — JavaScript foreign function interface (FFI) provided with Pyodide.

Pyjs:

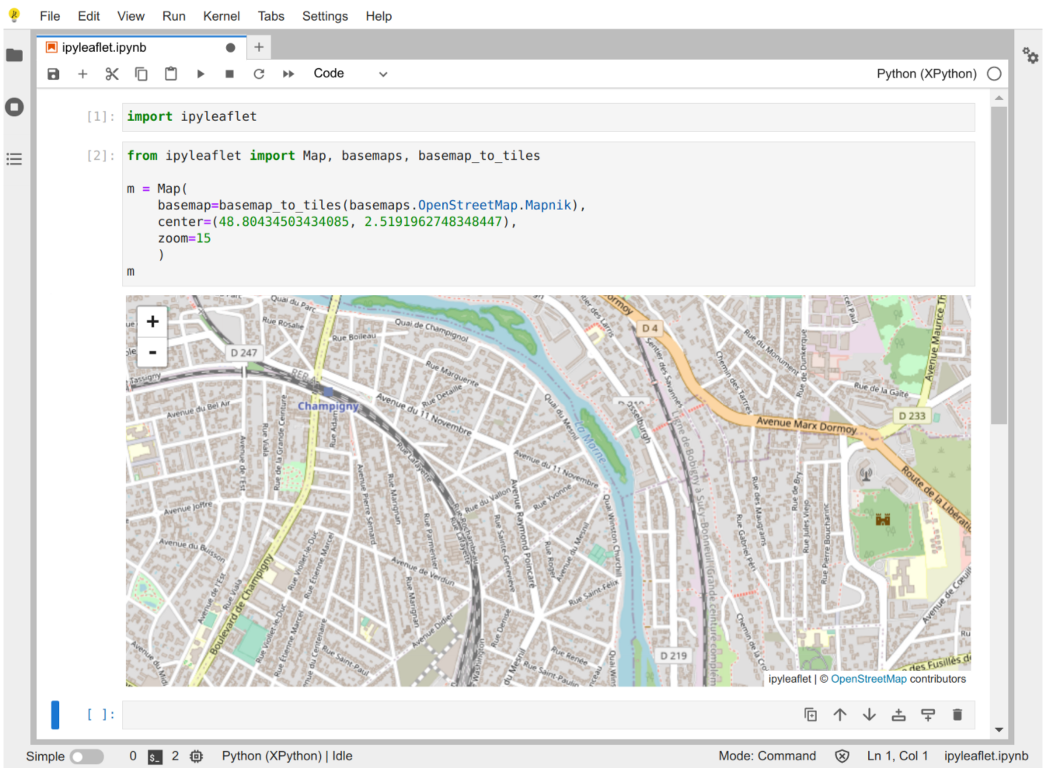
Since it is non-trivial to extract Pyodide's FFI and use it for other projects, we created a modern Python - JavaScript FFI from scratch. This was done with the following tricks:

- Pybind11 is used to call Python from C++ and vice versa,
- Embind is used to call JavaScript from C++ and vice versa.

When we use Pybind11 and Embind together we can call Python from JavaScript and vice versa, with C++ as a man in the middle. This not only allows us to write a simple FFI from scratch with relatively little code but also avoids calling any low-level CPython APIs and enables using high-level constructs — like `pybind11::object` and `emscripten::val` — instead.
The code is available in the pyjs repository. The API is very similar to Pyodide's so that it can be used as a drop-in replacement in code, like Pyodide's WebLoop implementation.

Deployment:
The xeus-python-kernel allows conda packages to be pre-installed in the Python

```
jupyter lite build --XeusPythonEnv.packages=\
    numpy,\
    matplotlib,\
    ipyleaflet
```


File Edit View Run Kernel Tabs Settings Help
ipyleaflet.ipynb
Code
Python (XPython)
ipyleaflet | © OpenStreetMap contributors
Simple 0 2 Python (XPython) | Idle Mode: Command Ln 1, Col 1 ipyleaflet.ipynb

```
[1]: import ipyleaflet

[2]: from ipyleaflet import Map, basemaps, basemap_to_tiles

     m = Map(
         basemap=basemap_to_tiles(basemaps.OpenStreetMap.Mapnik),
         center=(48.80434503434085, 2.5191962748348447),
         zoom=15
         )
     m

[ ]:
```

Running the xeus-python kernel with the ipyleaflet widget in JupyterLite

More details can be found in the xeus-python-kernel GitHub repository.

What about the future?

This combination of JupyterLite and Mamba has the potential to open Jupyter to millions of additional users.
Given its scalability, ease of deployment, reproducibility, and accessibility, JupyterLite will be everywhere: countries, organizations, and schools that don't have access to sovereign cloud infrastructure will be able to deploy Jupyter-based education platforms on servers that they truly own, without endangering the data of their students or becoming too reliant on resources that they do not control.

In the short term, we are working on the following "next steps":

- **Fortran:** Compiling Fortran code with Emscripten is currently not supported, but it is necessary for key packages like SciPy. Pyodide relies on f2c, a Fortran-to-C converter, in conjunction with a set of patches to compile Fortran code with Emscripten. We are working on a more direct approach: compiling SciPy natively with LFortran.
- **Binderlite:** Binder converts a repository of notebooks into an executable JupyterLab environment, making code immediately reproducible by anyone, anywhere. Emscripten-forge is the missing piece to build BinderLite, a version of Binder relying on JupyterLite instances instead of vanilla JupyterLab instances.
- **Rust/PyO3 support:** We are working on integrating the work of the Pyodide team on Rust/PyO3 support in emscripten-forge. This will be important to build Rust extension modules like `cryptography`.

Credits

This was built upon the work of a much bigger crowd!

The Pyodide team

The Pyodide project was started at the Mozilla foundation by Michael Droettboom and is now maintained by Hood Chatham, Roman Yurchak, and Gyeongjae Choi.

> The foundational work of the Pyodide project pioneered the use of Python in the browser and made all of the rest possible, from JupyterLite to this work.

The Emscripten team

Both Pyodide and emscripten-forge are built upon the Emscripten toolchain, which provides the foundational components to be able to meaningfully run WebAssembly programs in the browser.

The JupyterLite team

The JupyterLite project was started by Jeremy Tuloup, with significant contributions from Nick Bollweg and Martin Renou.

Boa tool to build them. In the mamba development team, we should highlight the work of Wolf Vollprecht, Johan Mabille, Joel Lamotte, and Andreas Trawöger.

The Xeus team

The xeus project was started by Johan Mabille and Sylvain Corlay. It is at the foundation of the JupyterLite integration and helped to get all the pieces together (Xeus, Mamba, Jupyter). We should especially credit the work of Martin Renou and Thorsten Beier on this integration with JupyterLite.

Acknowledgment

The work of Thorsten Beier, Johan Mabille, Martin Renou, Sylvain Corlay, Wolf Vollprecht, Joel Lamotte, and Andreas Trawoger at **QuantStack** was funded by **Bloomberg**.

About the Authors

Thorsten Beier

Thorsten Beier is a Scientific Software Engineer at QuantStack. Before joining QuantStack, he graduated in computer science at the University of Heidelberg and worked at the EMBL. As an open-source developer, Thorsten worked on a variety of projects, from xeus and xtensor in C++ to inferno, kipoi, ilastik, and napari-splineit in Python.

Martin Renou

Martin Renou is a Scientific Software Engineer at QuantStack. Before joining QuantStack, he studied at the French Aerospace Engineering School SUPAERO. He also worked at Logilab in Paris and Enthought in Cambridge. As an open-source developer at QuantStack, Martin worked on a variety of projects, from xsimd, xtensor, and xframe in C++ to ipyleaflet and ipywebrtc in Python and JavaScript.

Thanks to David Brochart, Sylvain Corlay, Wolf Vollprecht, and Jeremy Tuloup