Pyodide: scientific Python compiled to WebAssembly

Roman Yurchak

PyData Paris Meetup

June 3, 2021

PYODIDE

WA

Agenda

Pyodide overview

Applications

Current challenges and outlook

Pyodide overview

What is WebAssembly?

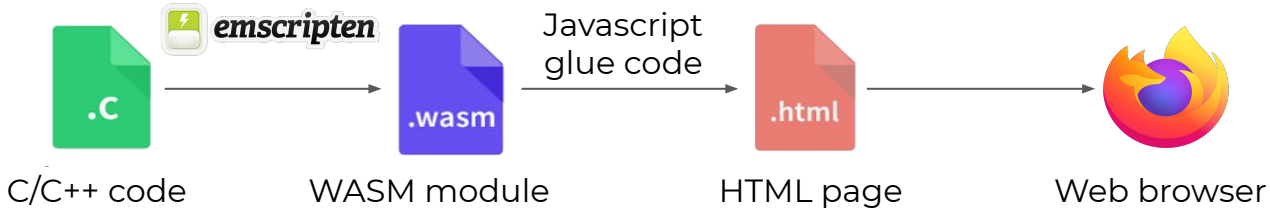
WebAssembly (or WASM) is a binary instruction format for a stack-based virtual machine,

- Initially implemented for browsers
- can also be executed in standalone environments (Node.js, WASI)

Features

- Portable (same binary for all OS / architectures)
- near native performance
- Sanboxed

Build pipeline



Pyodide project

CPython 3.8 and the scientific Python stack, compiled to WebAssembly

\+ micropip
Pure python wheels from PyPi

\+ Python / Javascript
Foreign Function Interface (FFI)

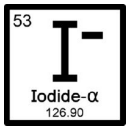
Project repository: github.com/pyodide/pyodide

PYODIDE
WA

Pyodide project history

Created by **Michael Droettboom** in 2018 at Mozilla.

Initially as a language plugin for **Iodide**, an experimental notebook environment for literate scientific computing and communication.



In 2021, Pyodide has become a independent and community driven project with,

- A governance
- A roadmap
- A code of conduct

More information: pyodide.org

PYODIDE
WA

Related projects

Several other projects also allow to run Python in the browser:

pypy.js

PyPy for Python 2 compiled
to asm.js
(No longer maintained)

Brython

Python 3 Javascript implementation
+ parts of stdlib

RustPython

A Python 3 interpreter written in Rust

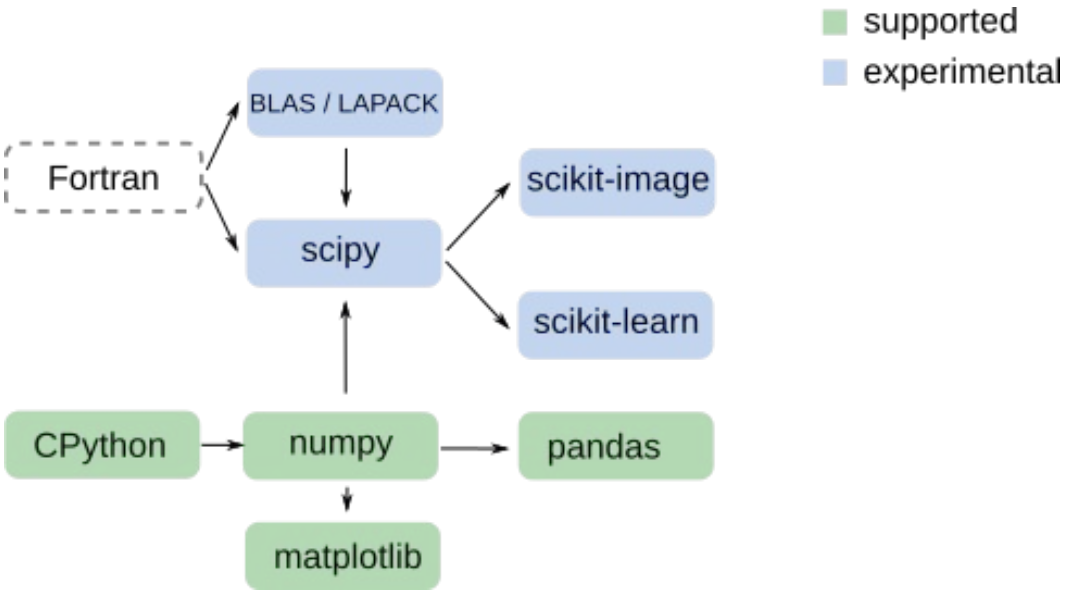
micropython-ports-wasm

A WASM MicroPython port (experimental)

Beyond the Python language, Pyodide aims to be compatible with the Python ecosystem.

Supported Python packages in Pyodide

with C extensions (examples)



\+

micropip

Pure python wheels from PyPi

Micropip and Python wheels

Pure Python packages

Installed with **micropip**, if wheels available:

- from PyPi or arbitrary location.
- rudimentary dependency resolution

Examples

See PEP 427:

-py3-none-any.whl -> pure Python wheel

-cp38-manylinux1_x86_64.whl -> Linux wheel (not compatible with pyodide)

Might still need to use the Pyodide build system, to apply patches (e.g. unsupported modules)

Python packages with C extensions

Need to use the Pyodide build system (write a meta.yaml, similar to conda-forge).

Distributed via JsDelivr CDN (.js & .data files). Can be loaded with pyodide.loadPackage (or micropip).

Python ↔ JavaScript type translations

Foreign function interface (FFI) between Python and JavaScript,

- convert to native types when possible (float, str, ..)
- otherwise proxy objects are used with a number of supported operators (getattr, setattr, __call__, ...)

Using Javascript from Python

Allows to access DOM or any JS object in the the global namespace

```
import js

js.document.title = "A new title"
```

Using Python from Javascript

A Python object (in global scope) can be brought over to Javascript

```
var sys = pyodide.globals.get('sys');
```

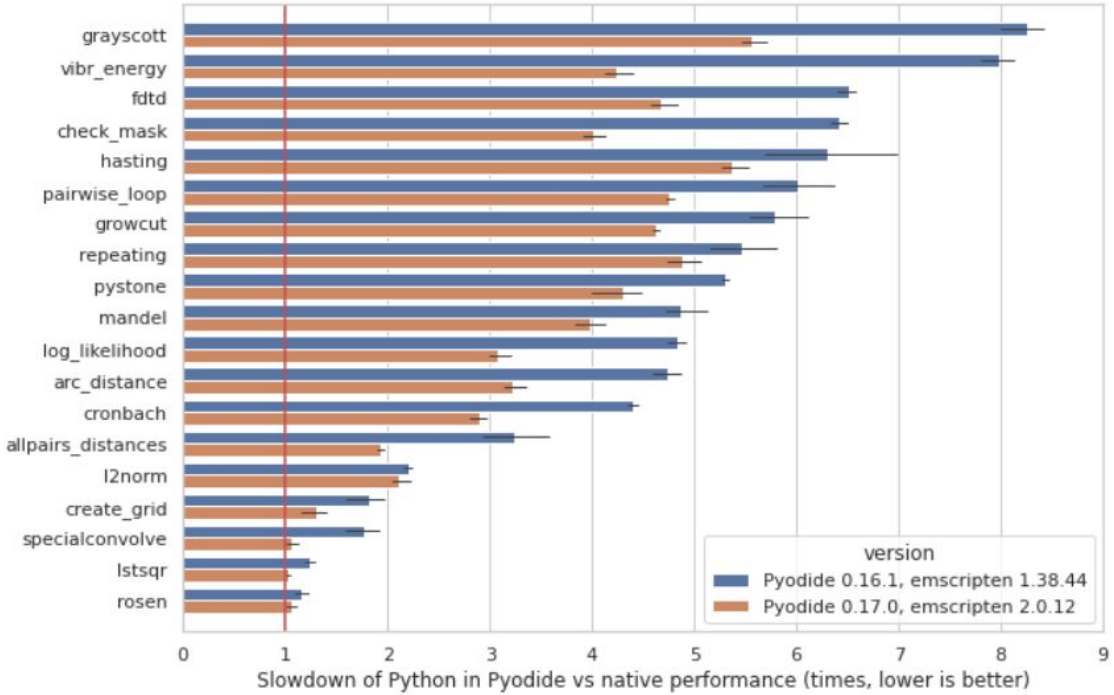
Fore more details: pyodide.org/en/stable/usage/type-conversions.html
Thanks to Hood Chatham for a major refactoring of type translations in 0.17.0

Demo

REPL

pyodide.org/en/stable/console.html

Performance



Up to 3-5x slower slower than native execution at present.
Lots of progress in the last Pyodide / emscripten releases.

Pyodide UIs and frontend Python apps

User Interfaces for pyodide (1)

Pyodide REPL

A simple JS console

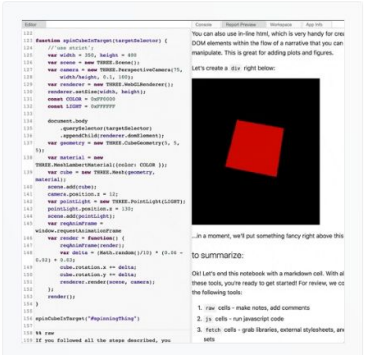
```
Welcome to the Pyodide terminal emulator
>>> from js import document
>>> document
[object HTMLDocument]
```

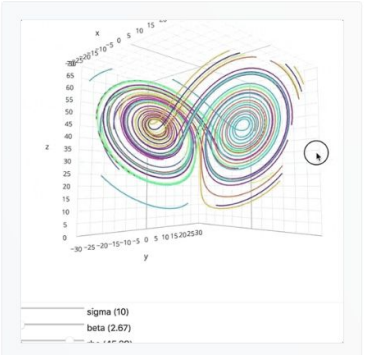
Iodide

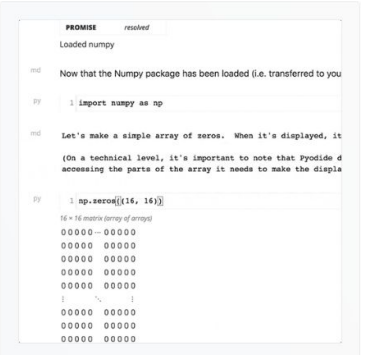
Literate scientific computing and communication for the web

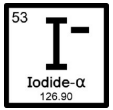
alpha.iodide.io

(no longer actively developed)





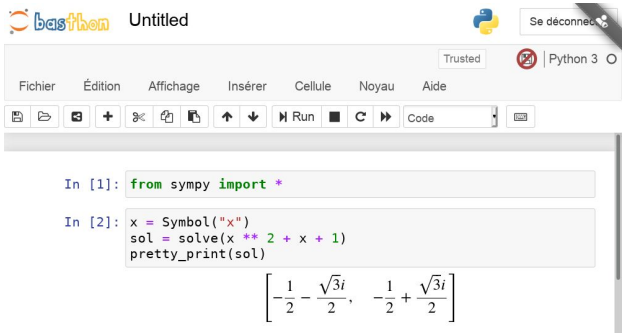




User Interfaces for Pyodide (2)

Basthon

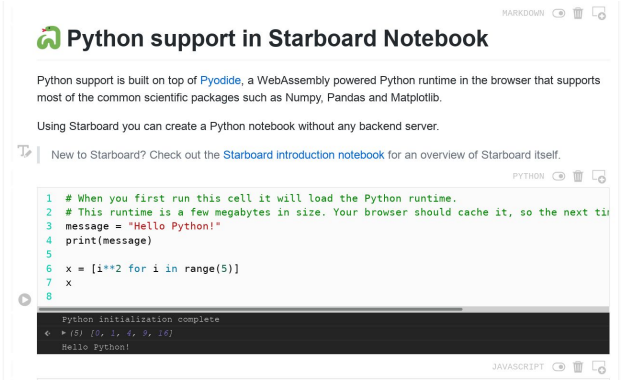
Static version of Jupyter notebook
notebook.basthon.fr
(currently in French)



basthon

Starboard Notebook

The shareable in-browser notebook
starboard.gg/#python



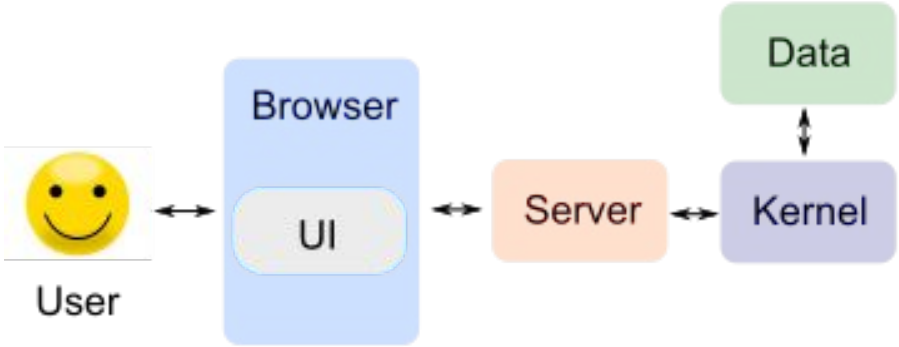
Starboard

JupyerLite

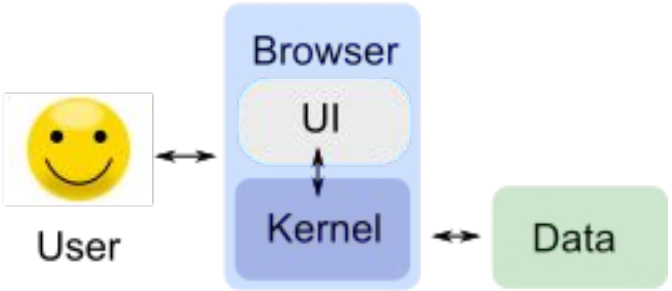
Wasm powered Jupyter github.com/jtpio/jupyterlite

Architecture

Application with a backend server



Application with only static files



Frontend web apps in Python

Usability

No Python installation needed, just open a web page

Scalability

Serving static files is easy, scales well to a large number of users

- No need for extensive backend infrastructure / maintenance effort
- *Example*: currently 50k-80k downloads/month for the main pyodide package

Packages only downloaded once, then cached in the browser

Frontend web apps in Python

Privacy

All calculations are run locally, no data needs to be sent to a remote server

Algorithmic Transparency

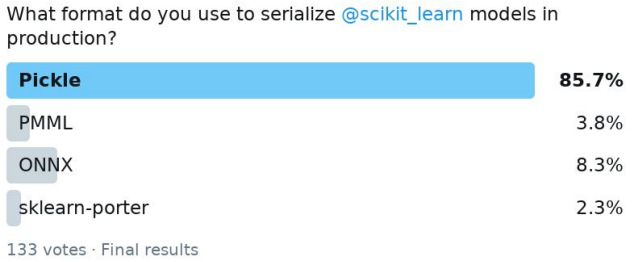
The app is just an archive with python files/objects

- can be examined on the client side
- can be either and advantage or a limitation depending on the use case

Deploying machine learning models

Classical workflow

1. Train the machine learning (ML) model
2. Serialize model to disk
3. Develop a web service
4. Package in a container (Docker)
5. Deploy on a server



Tools for ML inference with WASM support

ONNX

TensorFlow.js

tract

Fast, small model size but restricted to supported operators...

Deploying scikit-learn models in Pyodide*

Use pickle?

Unsafe, brittle to environment changes **but** portable and non opaque

Steps

1. Créate an environment with the same Python and dependencies versions as pyodide
2. Pickle the model (pickle.dumps) and deserialize it in Pyodide (pickle.loads)
3. Run inference from JS

Walk through: github.com/rth/notebooks/tree/master/pyodide/pyodide-ml-demo

* Experimental: see next part for the remaining challenges for use in production

Current challenges and roadmap

Download sizes for pyodide packages

Download size is not an optimisation criterion in the Python ecosystem (unlike for JS)

Large standard library (e.g. distutils 200kb)

Historically large packages (e.g. scipy)

- Possibly due the state of Python packaging 10+ years ago

Inclusions of test files in the main package (e.g. import numpy.tests)

- Also sometimes test data

Static imports analysis is challenging

- blanket imports in __init__, conditional imports

Package	Size	Size (brotli compression)
CPython/pyodide core	18 MB	6.3 MB
pandas	14.5 MB	7.6 MB

Significant optimization potential for the package sizes.

Known limitations

Some of the constraints that require workarounds,

WASM VM

- No subprocess, no threading (WIP)
- No sockets (HTTP requests need to be expressed with JS)
- 32 bit architecture
- Not all syscalls are implemented in emscripten

JavaScript

- No int type, only Number or BigInt
- No standard ND array type
- Can use reserved keywords from Python

```
Promise.new(...).then(...).finally(...)
```

SyntaxError in Python

Roadmap

- Smaller and faster to load packages
- Improve performance
- Re-implement some stdlib modules (e.g. http.client) with Web APIs, Sync I/O
- Updating scipy and Fortran support
- Sustainability of the package build system, upstreaming patches
- Threading (waiting for wider browser adoption)

See pyodide.org/en/stable/project/roadmap.html

New contributors are very welcome!

Both technical and non technical.

PYODIDE WA

Team and partners

Michael Droettboom
William Lachance
Brendan Colloran
Hamilton Ulmer
Teon Brooks

Hood Chatham
Dexter Chua
Roman Yurchak
Jan Max Meyer
Madhur Tandon

Romain Casati
Joe Marshall

Pyodide contributors

Partners and related projects

Thank you!

https://github.com/pyodide/pyodide

Questions ?

@RomanYurchak