Pyodide: scientific Python compiled in WebAssembly

Roman Yurchak

FOSDEM 2019
Iodide

An interactive programming environment for scientists in the browser

iodide.io

Examples

Lorenz Attractor

World Happiness Report

Eviction Notices in SF
Iodide overview

Using Python from Javascript

So far so good, but wouldn't it be great to use Python from within Javascript as well? When you load Pyodide, you'll get `pyodide` in the JS namespace, which lets you import anything on the Py side.

Pyodide brings the Python runtime to the browser via WebAssembly, along with NumPy, Pandas, Matplotlib, parts of SciPy, and NetworkX.

Press shift+enter to step through this notebook.

The real power of Pyodide comes from its scientific computing libraries. So far we've compiled numpy, pandas, matplotlib, parts of scipy, and networkx. At the top of any py chunk, simply write something like `import numpy as np` and run the chunk, and it'll begin the process of grabbing numpy.
Architecture

Jupyter-like model

User

Browser

UI

Server

Kernel

Data

Adapted from:
jupyter.readthedocs.io/en/latest/architecture/how_jupyter_ipython_work.html#notebooks
Architecture

Jupyter-like model

User → Browser → Server → Kernel → Data → Browser → User

Iodide

User → Browser → Kernel → Data → Browser → User

Adapted from:
jupyter.readthedocs.io/en/latest/architecture/how_jupyter_ipython_work.html#notebooks
Sharing of notebooks

Jupyter like model

Local

Install conda, Jupyter, then project-specific dependencies

Remote

Deploy in a container (binder etc.)
Sharing of notebooks

Jupyter like model

Local
Install conda, Jupyter, then project-specific dependencies

Remote
Deploy in a container (binder etc.)

Iodide model

Local
Deploy to a static webserver
Just open it in your browser

Remote
Share a single file containing data, report, code and dependencies
Just open it in your browser
Pyodide

Python scientific stack, compiled to WebAssembly

- created by Michael Droettboom
- language plugin for Iodide
- CPython interpreter
- numpy, pandas, matplotlib

WebAssembly

- A fast way to run compiled code in the browser

Related projects

- PyPy.js, brython, RustPython

[github.com/iodide-project/pyodide]
<html>
<head><meta charset="utf-8"/></head>

<body>
<script src="http://static.r0h.eu:59171/pyodide.js"></script>

<script>
languagePluginLoader.then(() => {
  pyodide.loadPackage(['numpy']).then(() => {
    pyodide.runPython(`
      import numpy as np

      x = np.random.rand(100)
      y = x.sum()
    `);

    var y = pyodide.pyimport('y');
    console.log(y);
  });});
</script>
</body>
</html>
Supported packages

CPython ➔ numpy ➔ pandas

- supported
- experimental
- planned
Supported packages

- CPython
- numpy
- pandas
- matplotlib
- Fortran
- scipy
- scikit-learn
- BLAS / LAPACK
- scikit-image

- supported
- experimental
- planned
Supported packages

- CPython
- numpy
- pandas
- matplotlib
- Fortran
- BLAS / LAPACK
- scipy
- scikit-learn
- scikit-image

- Pure python wheels on PyPi

supported
experimental
planned
Performance

Firefox: 4-8 slower for pure Python, 1-2 times slower for C-ext. Ideal scaling with the number of users.

github.com/iodide-project/pyodide/tree/master/benchmark
Build process

CPython interpreter → C extensions → Python files → Emscripten compiler / toolchain → .wasm → .wasm → .py → Browser engine

In memory filesystem, lz4 compressed

emscripten.org
System calls

For example,

- \(\downarrow \text{os.open} \) in Python
- \(\downarrow \text{CPython: call os_open_impl C function} \)
System calls

For example,

- ↓ `os.open` in Python
- ↓ CPython: call `os_open_impl` C function

Linux

- ↓ `open` system call to glibc
- ↓ Linux kernel
System calls

For example,

- ↓ os.open in Python
- ↓ CPython: call os_open_impl C function

Linux

- ↓ open system call to glibc
- ↓ Linux kernel

Emscripten / WebAssembly

- ↓ Emscripten
- ↓ system call to musl libc
- ↓ WebAssembly engine
System calls (sometimes)

For example,

- \texttt{os.statvfs} in Python (disk space usage)
- CPython: call \texttt{os\_statvfs\_impl} C function

**Linux**

- \texttt{statvfs} system call to glibc
- Linux kernel

**Emscripten / WebAssembly**

- Emscripten: not implemented; return "safe and sane values"

  - system call to musl libc
  - X WebAssembly engine

Most system calls work, but there are some edge cases.
What doesn't work

Difficult

- network sockets
- multiprocessing
- host filesystem access

Should work someday

- threads
- async
Testing

Pytest is supported: test collection and execution in the browser
Testing

Pytest is supported: test collection and execution in the browser

Test suites

- CPython: 380 test files / 497 pass (increasing, but some will never pass due to WebAssembly environment)
- numpy: 3145 passed, 42 failed (+ some collection failures), 47 skipped
- scikit-learn: WIP, looks promising. Some remaining issues with Fortran / LAPACK calls in scipy.
Planned language interoperability
Future work

• increase the percentage of passing tests
• dynamic linking of BLAS/LAPACK in scipy
  ◦ possible in Emscripten 1.38.22 thanks to Kirill Smelkov
• optimize download sizes
• threading and async support
• more packages

Contributors welcome!
Application: in-browser data analytics

- challenges of multi-user notebooks deployment
- running notebooks on the edge with uncertain/limited connectivity

- Iodide and Pyodide integrated into the OfficeJS apps store
  - online / offline usage, synchronization in Dropbox etc
Development team

Brendan Colloran
Hamilton Ulmer
William Lachance
Michael Droettboom
Teon Brooks
...

Mozilla

nexedi
Thank you!

Questions?

github.com/iodide-project/pyodide  @RomanYurchak