# Pyodide: a Python distribution for the browser

Hood Chatham, Roman Yurchak


# About us

## Hood

UCLA “NSF Assistant Adjunct Professor”
i.e., a post doc

Partially supported by NSF grant #DMS–2002087

## Roman Yurchak

Data scientist
Core developer at scikit-learn, Pyodide

@RomanYurchak

SYMERIO

# Agenda

1. What is Pyodide?
2. Use cases: interactive computing, education, ML
3. Latest developments and outlook

# Serverless Python apps for the web?

**Problem**

Python web apps are complicated:

- Frontend / JavaScript code
- Backend / Python code
- Infrastructure: server maintenance or cloud configuration

**Goal**

What if we could take a Python application, and run the code directly in the browser?

Now possible with WebAssembly!

*See Peter Wang's keynote at PyCon US this morning.*

# Python in the browser with Pyodide: an overview

# What is WebAssembly?


A binary instruction format for a stack-based virtual machine

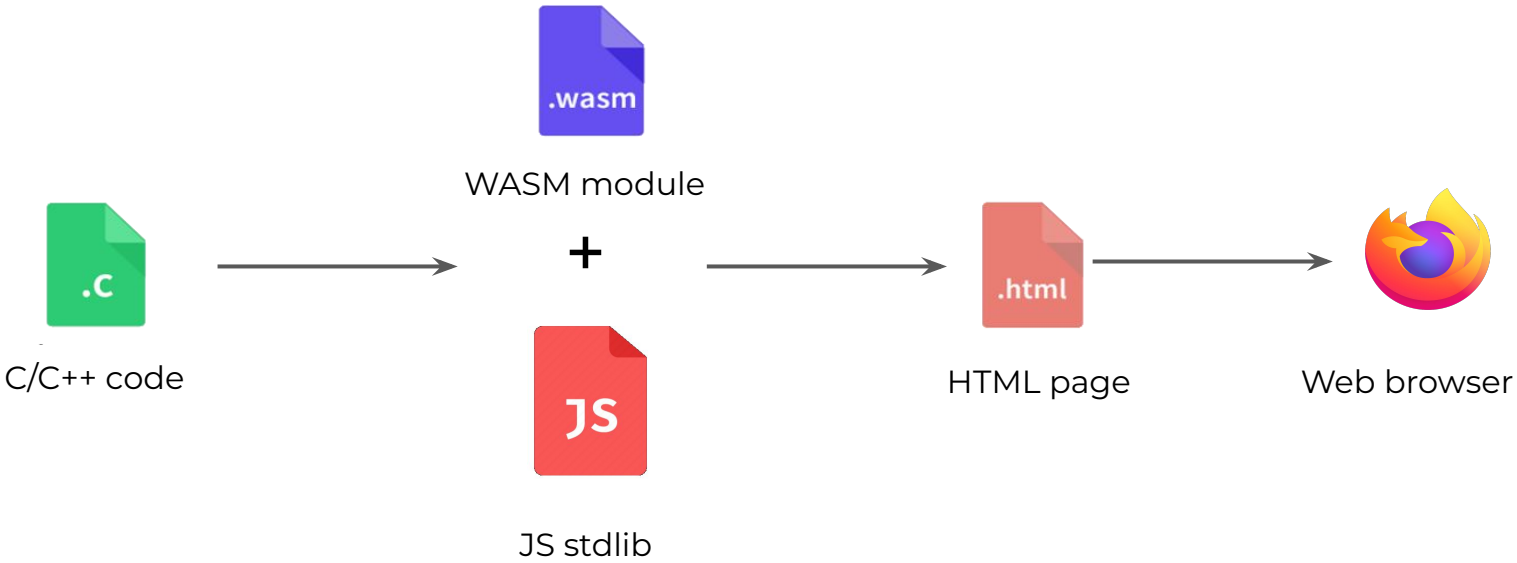
- Portable
- Small code size
- Secure
- No standard APIs or syscalls, only an import mechanism

- Implemented in browsers
- Can also be executed in non-web environments

https://webassembly.org/

# The emscripten build toolchain

**Emscripten is a complete compiler toolchain targeting WebAssembly**




https://emscripten.org/

# Pyodide Components

CPython

\+

Python / Javascript
Foreign function interface

emscripten

WASM +
Javascript stdlib

NumPy pandas

SciPy matplotlib

...

\+

micropip
Pure python wheels
from PyPi

Pyodide was created by Michael Droettboom at Mozilla

PYODIDE
WA

# Upstream CPython WASM work

Since 2018 Pyodide was building CPython with many patches.

In 2022 work started on adding WASM build targets in CPython upstream.

Lots of improvements and fixes in Python 3.11.a7+ (cpython#84461)

- Upstreaming of Pyodide patches
- Contributing Emscripten fixes
- More of CPython test suite passes
- Planned Tier 3 support

See Christian Heimes' keynote at PyConDE

Thanks to Christian Heimes, Brett Cannon, and Ethan Smith.

This will make Pyodide more sustainable.

# Related projects

A number of other projects also allow to run Python in the browser:

- **Brython**: Python 3 javascript implementation + parts of the stdlib
- **pypy.js**: PyPy compiled to asm.js  (no longer maintained)
-  **RustPython**: using the Rust toolchain to build for WASM

For practical usage, compatibility and access to the package ecosystem is critical.

# Pure Python packages with micropip

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

- from PyPI or arbitrary location
- rudimentary dependency resolution

Some packages need to be patched,
- with ongoing effort to upstream fixes

**Examples**

See PEP 427:

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

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

# Packages with binary extensions

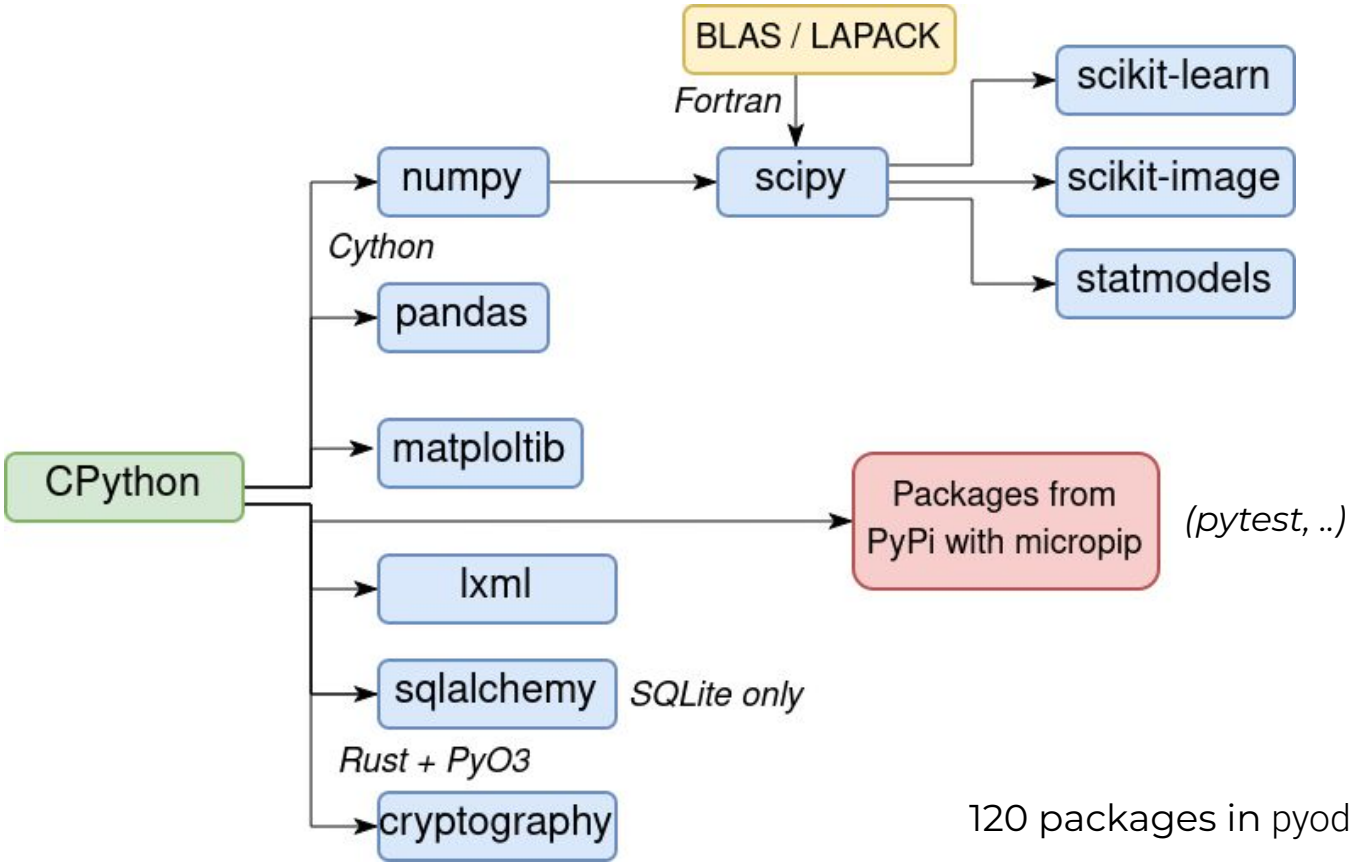
Need to use the Pyodide build system (write a **meta.yaml**, similar to conda)

- A cross-compilation setup, now building wheels
- Recent support for **pypa/build** for build isolation
- Additional post-processing: unvendoring tests as separate packages
- Still a long way to a wheel standard for WASM, before their support on PyPI
  - No stable ABI in Emscripten

Wheels distributed via  JSDelivr.

There are also other more conda / conda-forge oriented initiatives (**emscripten-forge**).

# Supported Python packages in Pyodide




120 packages in pyodide/packages/ ...

# Foreign function interface (JS ↔ Python)

**Using Javascript from Python**

A Javascript object in global scope can be imported into Python

```
from js import setTimeout

setTimeout(f, 100)
```

**Using Python from Javascript**

A Python object in global scope can be accessed from Javascript

```
let sum = pyodide.globals.get("sum");

sum([1, 3, 4]); // 8
```

- Automatic conversion of simple native types (float, str, int, ...,)
- Other types are proxied

For more details: [pyodide.org/en/stable/usage/type-conversions.html](pyodide.org/en/stable/usage/type-conversions.html)

# Example: Python utils from JavaScript

```
const functools =
pyodide.pyimport("functools");
functools.reduce((x,y) => x*y, [1,2,3,4]);
```

```
const math = pyodide.pyimport("math");
math.lcm(4, 6, 13); # Least common multiple
```

# Example: random.sample

From Javascript:

```
const random = pyodide.pyimport("random");
random.sample(
    pyodide.toPy(['red', 'blue']),
    5
).toJs();
```

# Example: random.sample

A Python random.sample wrapper for use from Javascript:

```
def random_sample_from_js(space, n):
    from pyodide import to_js
    from random import sample

    space_py = space.to_py()
    result = sample(space_py, n)
    return to_js(result)
```

# Examples: fetch API from Python

```
from js import fetch

response = await fetch("example.com", method="GET",
redirect="error")
text = await response.text()
```

# Examples: Buffers

Can use numpy arrays with Javascript ndarray libraries (e.g., video processing)

```
function editBuffer(x, idx){
    let buf = x.getBuffer();
    buf.data[idx] *= 3;
    buf.release();
}

pyodide.runPython(`
    from js import editBuffer
    ar = np.arange(10, dtype=np.float)
    editBuffer(ar, 3)
`);
```

# Emscripten Host Environment

**Features**

- 32 bit architecture
- (Javascript) Memory Filesystem
- System calls implemented in Javascript

**Limitations**

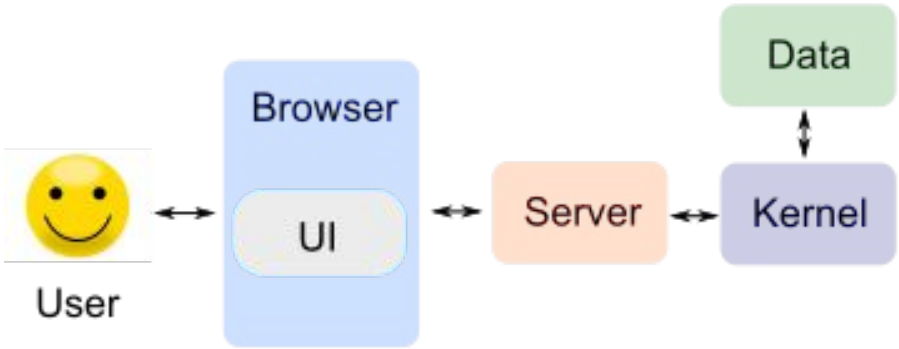
- No subprocess, no threading (theoretically possible, significant work needed)
- No sockets
- Not all syscalls are implemented in Emscripten
- Difficult to use traditional I/O

# Some use cases

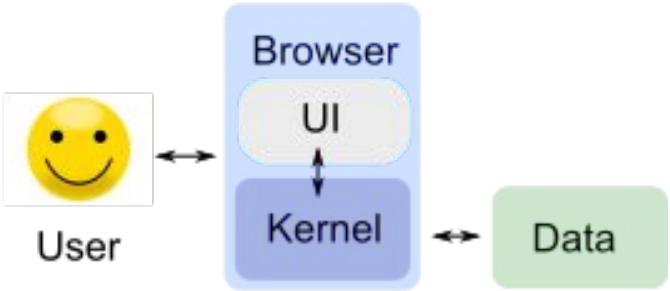
Interactive computing
Education
Machine learning

# Client-only Architecture

Application with a backend server




Application with only static files



# Client-only 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

Packages only downloaded once, then cached in the browser

# Client-only Web Apps in Python

## Privacy

**All calculations run locally, no data sent to a remote server**

- Good for users
- Good for developers (less GDPR related paperwork)

See: “Analyzing sensitive data at scale doesn’t have to be a headache” by Tambe Tabitha

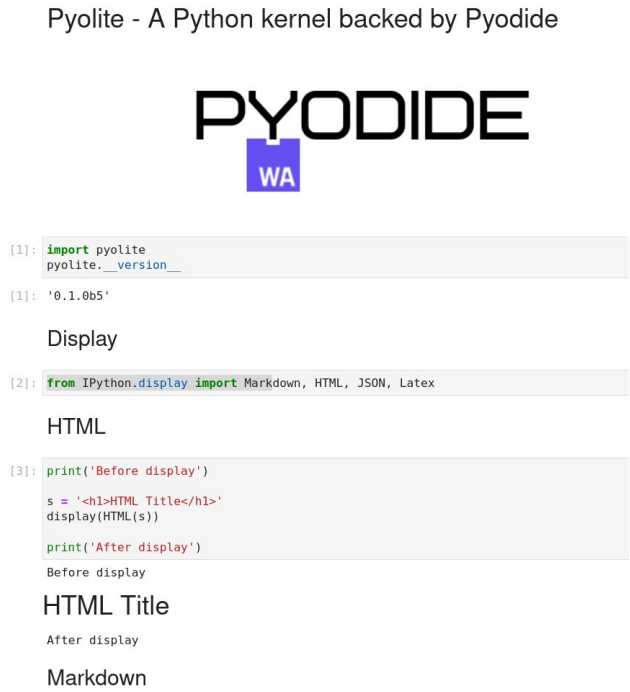
www.socialfinance.org.uk/blogs/analysing-sensitive-data-scale-doesn’t-have-be-headache

# A growing ecosystem

- **Pyscript**: a framework to create rich Python applications in the browser using HTML <py> pyscript.net/ (see Peter Wang's keynote at PyCon US)
- **Irydium**: Interactive documents and data visualizations in markdown irydium.dev
- **React + Pyodide**: using a JavaScript framework in Python blog.pyodide.org/posts/react-in-python-with-pyodide/
- **wc-code**: running Python code snippets with HTML tags github.com/vanillawc/wc-code

# Notebook environments


jupyterlite.readthedocs.io




Many other interactive computing projects:

- **Starboard Notebook**: The shareable in-browser notebook starboard.gg/#python
- **Basthon**: Static version of Jupyter notebook notebook.basthon.fr (in French)

# Pyodide in Education

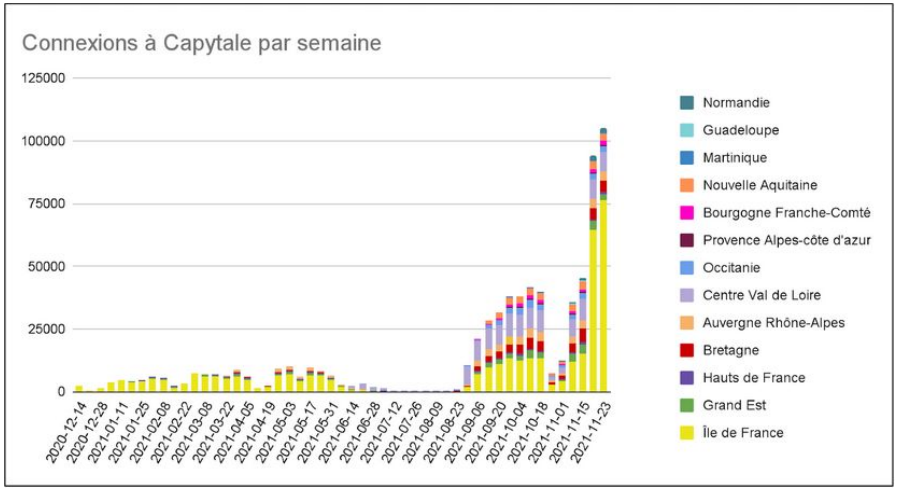
Python is taught in French high schools:

- Python installation is time consuming and teachers can't spend time on it
- A centralized server infrastructure to run code is costly

A notebook solution (Basthon) backed by Pyodide is now used in as part of the Capytale project.

100k+ weekly users with a minimal effort of serving static files.

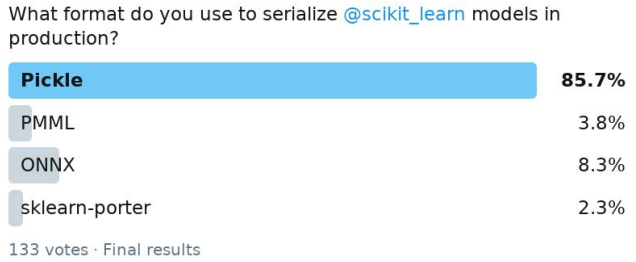
Many other efforts for education and research: futurecoder, EngineeringPaper.xyz, ..



# 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 predefined operators...

# Deploying scikit-learn models in Pyodide

## Use pickle?

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

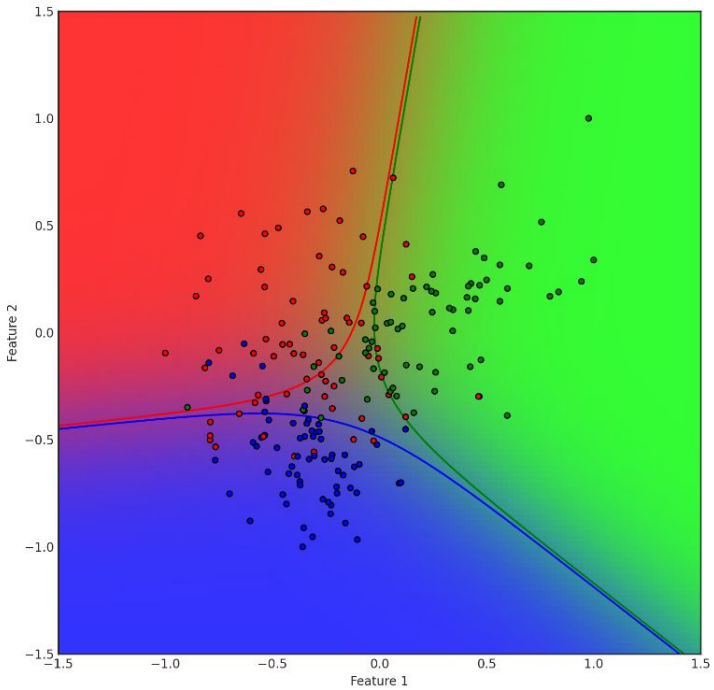
## Steps

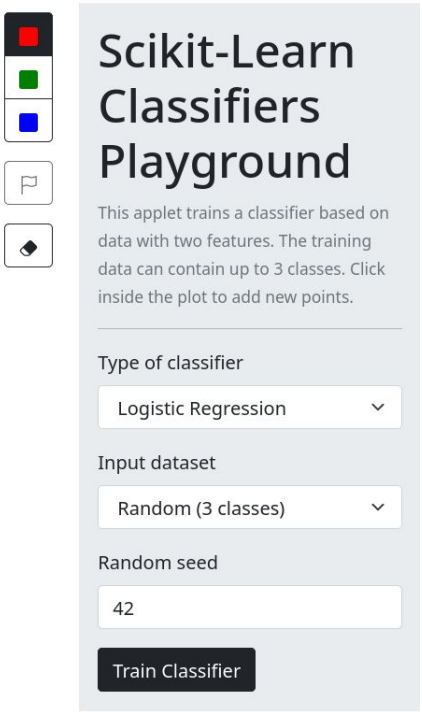
1. Create an environment with the same Python and dependencies versions as Pyodide
2. Pickle the model (pickle.dumps) and deserialize it in pyodide (pickle.loads)
3. Run inference from JS

Training can also happen directly on the client.

# Classifier decision boundary example

A React app to train scikit-learn classifiers online, using synthetic datasets by Stefano Meschiari




www.stefanom.io/sklearn-classifiers-playground

# Latest developments and outlook

# Packaging SciPy and Fortran

There is no working Fortran compiler with based on LLVM with WASM support

- WIP: LFortran, Flang classic, Flang
- gcc plugin ⇒ LLVM IR from gfortran

Instead we use f2c...

> *"f2c is a program to convert Fortran 77 to C code, developed at Bell Laboratories"*

but f2c *also* doesn't work for us

We use a mixture of automatic Fortran source transformations, automatic C transformations, and manual patches... =(

# Function Pointer Cast Handling

- Python C extensions define Python functions in C, but with the wrong number of arguments.
- The C standard says this is undefined behavior,
  most C compilers generate correct code.
- WASM checks the signature of function pointers when it calls them

```
call_indirect (i32, i32) -> i32 function_ptr
```

# Function Pointer Cast Handling

```
// C extension
PyObject* do_something(PyObject *self)
{
  // ... do stuff
  PY_RETURN_NONE;
}
PyMethodDef do_something_def = {
  "do_something", // the name
  (PyCFunction)do_something, // Function pointer cast!!
  METH_NOARGS, // the calling convention
};
```

---

```
// Called from the interpreter in methodobject.c:
PyObject *result = meth(PyCFunction_GET_SELF(func),NULL);
```

# Function Pointer Cast Handling

- Python C extensions define Python functions in C, but with the wrong number of arguments.
- The C standard says this is undefined behavior,
  most C compilers generate correct code.
- WASM checks the signature of function pointers when it calls them

  call_indirect (i32, i32) -> i32 function_ptr

- Javascript ⇒ Wasm calls are flexible
  so use a trampoline call Wasm ⇒ Javascript ⇒ Wasm
- upstreamed into Python 3.11! [cpython#32189]

Detailed discussion: [blog.pyodide.org/posts/function-pointer-cast-handling/]

# Getting http.client to work (WIP)

Sockets don't work in WASM VM ⇒ use fetch

**Problem**

Synchronous C / Python APIs wish to consume asynchronous browser APIs.

**Solution**

- Run Pyodide in a Worker (browser version of processes),
- send requests to a separate thread,
- use Atomics API to block for thread to complete

WIP: github.com/hoodmane/synclink

**Example**

- Can use Chrome FileSystem API to mount a native directory into Pyodide
- Apps can write directly into user's host OS file system

github.com/hoodmane/worker-pyodide-console/tree/nativefs

# Asyncio in the browser

Each browser thread comes with an event loop.

**Pyodide has WebLoop**

- Schedule tasks on the browser event loop

**Limitations**

- We cannot block ⇒ asyncio.run_until_complete cannot work as expected
- No control over event loop lifecycle

**Benefits**

- No need to control event loop lifecycle!

# Download sizes for packages

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

Historically large packages (e.g. scipy)

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

*Example of loading pandas*

| | | | | | |
|---|---|---|---|---|---|
| 200 | GET | distutils.tar | x-tar | 961.10 KB | 960 KB |
| 200 | GET | favicon.ico | x-icon | 667 B | 766 B |
| 200 | GET | jquery | js | 32.36 KB | 87.40 KB |
| 200 | GET | jquery.terminal.min.css | css | 5.30 KB | 22.83 KB |
| 200 | GET | jquery.terminal.min.js | js | 54.38 KB | 162.60 KB |
| 200 | GET | numpy-1.22.3-cp310-cp310-emscripten_wasm32.whl | octet-... | 3.53 MB | 3.53 MB |
| 200 | GET | packages.json | json | 5.86 KB | 27.39 KB |
| 200 | GET | pandas-1.4.2-cp310-cp310-emscripten_wasm32.whl | octet-... | 4.97 MB | 4.97 MB |
| 200 | GET | pyodide.asm.data | wasm | 3.21 MB | 5.14 MB |
| 200 | GET | pyodide.asm.js | js | 315.25 KB | 1.91 MB |
| 200 | GET | pyodide.asm.wasm | wasm | 3.04 MB | 9.05 MB |
| 200 | GET | pyodide.js | js | 14.75 KB | 44.92 KB |
| 200 | GET | pyodide_py.tar | x-tar | 101.13 KB | 100 KB |
| 200 | GET | pyparsing-3.0.7-py3-none-any.whl | octet-... | 96.89 KB | 95.75 KB |
| 200 | GET | python_dateutil-2.8.2-py2.py3-none-any.whl | octet-... | 243.04 KB | 241.90 KB |
| 200 | GET | pytz-2022.1-py2.py3-none-any.whl | octet-... | 492.86 KB | 491.72 KB |
| 200 | GET | setuptools-62.0.0-py3-none-any.whl | octet-... | 773.12 KB | 771.98 KB |
| 200 | GET | six-1.16.0-py2.py3-none-any.whl | octet-... | 11.93 KB | 10.79 KB |

20 requests | 27.56 MB / 17.79 MB transferred | Finish: 14.27 s | DOMContentLoaded: 18

# Make Python package sizes web friendly

**Break large packages in smaller parts**

- Makes it difficult to reuse existing dependency lists

**Use a bundler tool**

- Detect modules used at runtime, create a separate archive with those
- The code to run needs be known in advance

**Dynamic imports**

- Fetch Python modules as they are loaded
- performance concerns

**Wait for the average web page size to grow larger** (1 MB in 2012, 2 MB in 2018) ...

# Roadmap

- Keep up with Emscripten releases (fixes, size and performance improvements)
- Upstream package patches (Numpy)
- Support for synchronous I/O and web workers
- Reimplement some stdlib modules (e.g. `http.client`) with Web APIs
- Reduce size of packages
- Improve sustainability of the package build system
- Threading?

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

Also much exciting work to be done upstream and downstream

New contributors are very welcome!

Many low hanging fruit in the Python for WASM ecosystem.

PYODIDE WA

# Acknowledgement


## Pyodide project

Michael Droettboom

Gyeongjae Choi

Joe Marshal

Henry Schreiner

Dexter Chua

Pyodide commiters and users who engaged in discussions on the issue tracker.

Pyodide sponsors 

## Community

**Emscripten**

Alon Zakai and Sam Clegg

**CPython**

Christian Heimes, Brett Cannon, Ethan Smith

**JupyterLite, Basthon, pyscript, Irydium maintainers**

**Iodide team** Brendan Colloran, Hamilton Ulmer, Will Lachance

Python package maintainers for reviewing patches to improve Pyodide compatibility


PYODIDE WA

Thank you!

github.com/pyodide/pyodide

Join us at the PyCon sprints

@pyodide

roberthoodchatham@gmail.com

@RomanYurchak