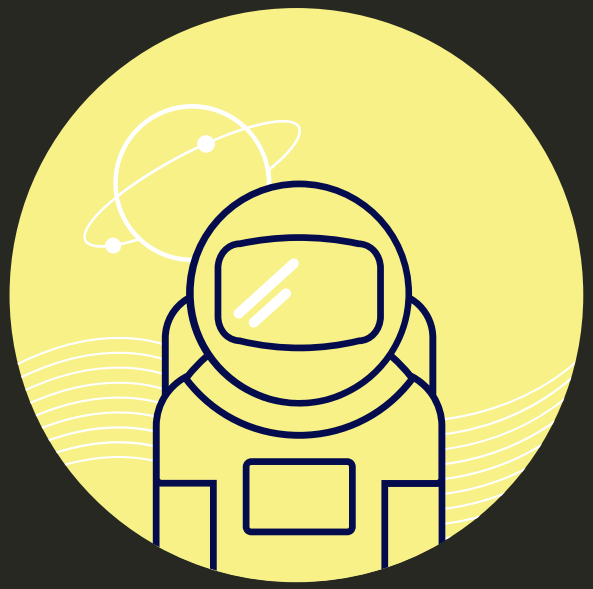
emscripten-forge

build wasm with conda

Thorsten Beier

QuantStack

2022



JupyterLite

JupyterLite is a JupyterLab distribution that runs entirely in the browser

JupyterLite

JupyterLite

https://jupyterlite.readthedocs.io/en/latest/_static/lab/index.html

File Edit View Run Kernel Tabs Settings Help

Filter files by name

/

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

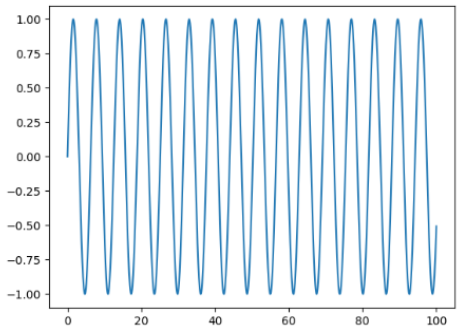
pyolite.ipynb

Markdown Pyolite

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
```

Console 1

```
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.ipynb

Markdown p5.js

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;
```

Simple 0 3 Pyolite | Idle Ln 1, Col 1 Console 1

JupyterLite runs entirely in the browser
$\Rightarrow$ JupyterLite kernels need to run in the browser

- JS JavaScript kernels:
  - JS JavaScript
  - p5* P5

JupyterLite runs entirely in the browser
$\Rightarrow$ JupyterLite kernels need to run in the browser

- JS JavaScript kernels:
  - JS JavaScript
  - p5* P5
- WA WebAssembly kernels:

JupyterLite runs entirely in the browser
⇒ JupyterLite kernels need to run in the browser

- JS JavaScript kernels:
  - JS JavaScript
  - p5* P5
- WA WebAssembly kernels:
  - Xeus kernels:
    - Xeus Lua lua
    - Xeus sqlite
    - Xeus wren

JupyterLite runs entirely in the browser
⇒ JupyterLite kernels need to run in the browser

- JS JavaScript kernels:
  - JS JavaScript
  - p5* P5
- WA WebAssembly kernels:
  - Xeus kernels:
    - Xeus Lua lua
    - Xeus sqlite
    - Xeus wren
  - PYODIDE pyodide

Pyodide

a Python distribution for the browser compiled with emscripten

- A lot of packages are available:
  - Micropip for pure Python packages:

    ```
    micropip.install('six')
    ```

  - $\sim$ 100 compiled packages: numpy, scipy, ...
- JavaScript $\Leftrightarrow$ Python

- Odd build system
- Only single version of each package
- Does not utilize conda

Building WebAssembly

Build WebAssembly with Emscripten

How to get started

```
#include <iostream>
#include <emscripten/val.h>
#include <emscripten/bind.h>

void hello_world()
{
   std::cout<<"hello world\n";
}

EMSCRIPTEN_BINDINGS(my_module) {
    emscripten::function("hello_world", &hello_world);
}
```

Build WebAssembly with Emscripten

How to get started

```
emcmake cmake . \
 -DCMAKE_INSTALL_PREFIX=$HOME/wasm_install \
 -DCMAKE_PREFIX_PATH=$HOME/wasm_install
emmake make -j12
emmake make install
```

Build WebAssembly with Emscripten

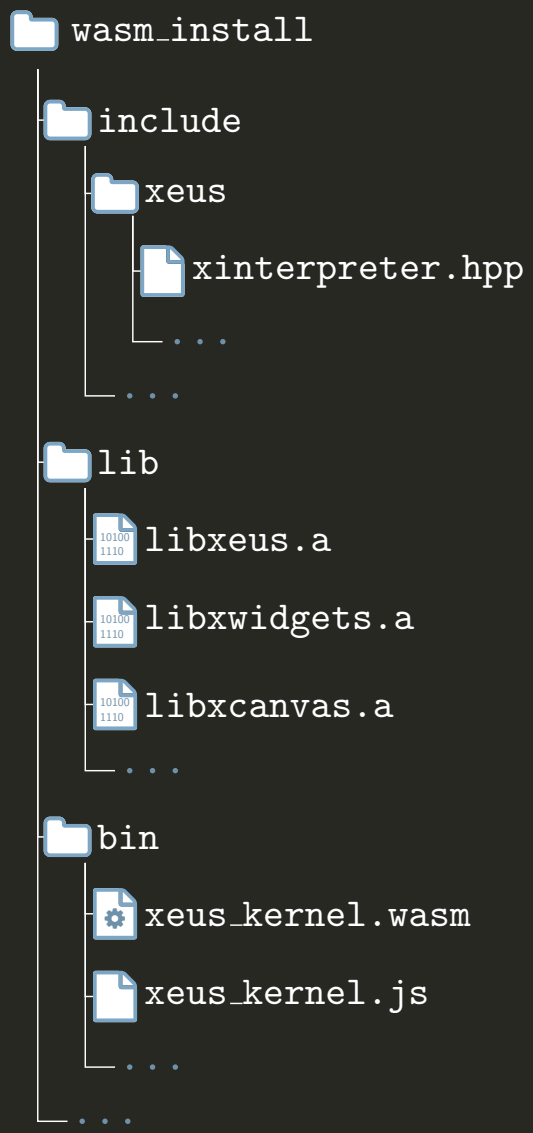
Things escalate quickly

```
# xproperties
cd xproperties
emcmake cmake . \
 -DCMAKE_INSTALL_PREFIX=$HOME/wasm_install \
 -DCMAKE_PREFIX_PATH=$HOME/wasm_install
emmake make -j12 install

# xwidgets
cd $HOME/src/xwidgets
emcmake cmake . \
 -DCMAKE_INSTALL_PREFIX=$HOME/wasm_install \
 -DCMAKE_PREFIX_PATH=$HOME/wasm_install
emmake make -j12 install

# xcanvas
cd $HOME/src/xcanvas
emcmake cmake . \
 -DCMAKE_INSTALL_PREFIX=$HOME/wasm_install \
 -DCMAKE_PREFIX_PATH=$HOME/wasm_install
emmake make -j12 install

# xcanvas
...
```

Build WebAssembly with Emscripten

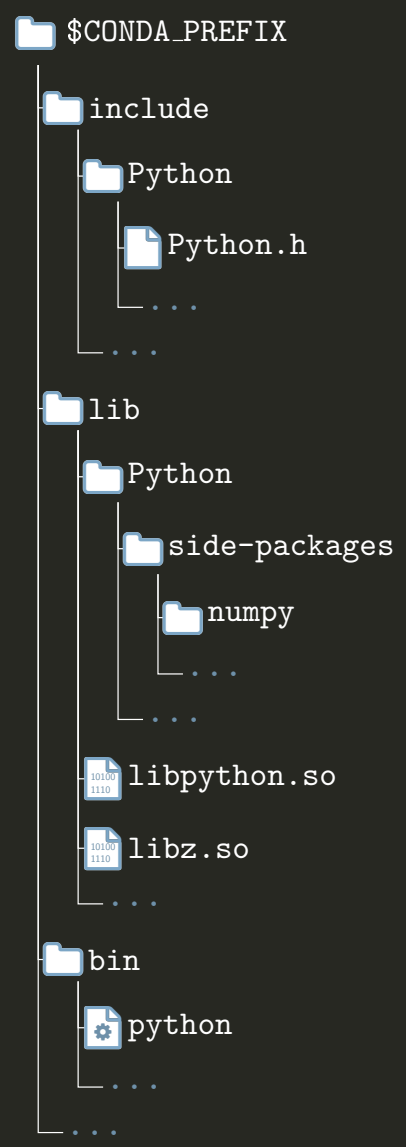
Things escalate quickly

```
# xproperties
cd xproperties
emcmake cmake . \
 -DCMAKE_INSTALL_PREFIX=$HOME/wasm_install \
 -DCMAKE_PREFIX_PATH=$HOME/wasm_install
emmake make -j12 install

# xwidgets
cd $HOME/src/xwidgets
emcmake cmake . \
 -DCMAKE_INSTALL_PREFIX=$HOME/wasm_install \
 -DCMAKE_PREFIX_PATH=$HOME/wasm_install
emmake make -j12 install

# xcanvas
cd $HOME/src/xcanvas
emcmake cmake . \
 -DCMAKE_INSTALL_PREFIX=$HOME/wasm_install \
 -DCMAKE_PREFIX_PATH=$HOME/wasm_install
emmake make -j12 install

# xcanvas
...
```



CONDA

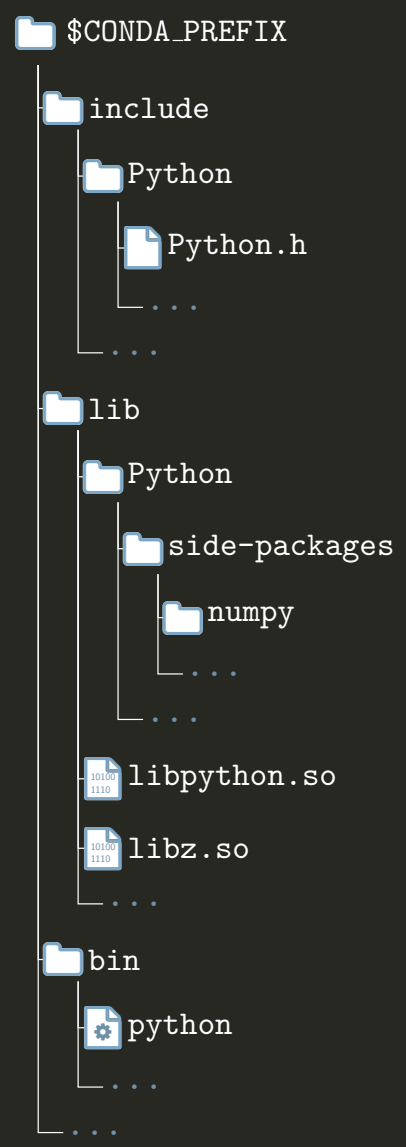
conda and wasm



Conda:

- Perfect for managing environments / prefixes

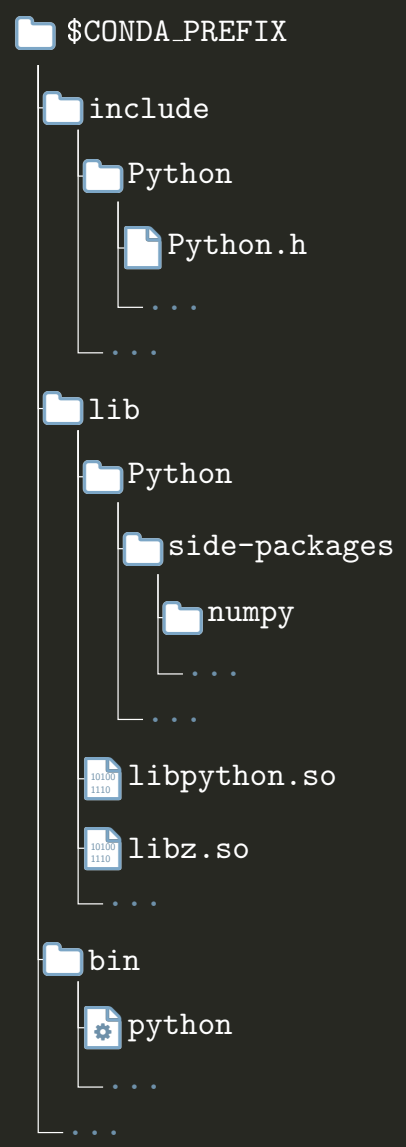
conda and wasm



Conda:

- Perfect for managing environments / prefixes
- Has cross compiling capabilities

conda and wasm



Conda:

- Perfect for managing environments / prefixes
- Has cross compiling capabilities
- We *just* need to add emscripten / wasm support to conda

- ☐ Add *emscripten32* as plattform to { conda, conda-build }
- ☐ Add compiler package for *emscripten32*

conda-build – metadata.py

```python
def ns_cfg(config):
    plat = config.host_subdir
    d = dict(
        linux=plat.startswith('linux-'),
        linux32=bool(plat == 'linux-32'),
        linux64=bool(plat == 'linux-64'),
        arm=plat.startswith('linux-arm'),
        osx=plat.startswith('osx-'),
        emscripten=plat.startswith('emscripten-'),
        emscripten32=bool(plat == 'emscripten-32'),
        unix=plat.startswith(('linux-', 'osx-', 'emscripten-')),
        win=plat.startswith('win-'),
        win32=bool(plat == 'win-32'),
        win64=bool(plat == 'win-64'),
        x86=plat.endswith(('-32', '-64')),
        x86_64=plat.endswith('-64'),
        os=os,
        environ=os.environ,
        nomkl=bool(int(os.environ.get('FEATURE_NOMKL', False)))
    )
```

conda-forge-pinning-feedstock – conda_build_config.yaml

```yaml
cross_compiler_target_platform: # [win]
  - win-64                      # [win]
c_compiler:
  - gcc                         # [linux]
  - clang                       # [osx]
  - vs2017                      # [win]
  - emscripten                  # [emscripten]
c_compiler_version:             # [unix]
  - 11                          # [osx]
  - 9                           # [linux]
  - 3                           # [emscripten]
cxx_compiler:
  - gxx                         # [linux]
  - clangxx                     # [osx]
  - vs2017                      # [win]
  - emscripten                  # [emscripten]

...
...
```

conda and wasm

The emscripten compiler package – recipe.yaml

```yaml
context:
  name: emscripten_emscripten-32
  version: 3.1.2

package:
  name: '{{ name|lower }}'
  version: '{{ version }}'

build:
  number: 0

about:
  home: emscripten
  license: BSD-3-Clause
  license_family: BSD
  license_file: LICENSE
  summary: emscripten
  description: emscripten
  doc_url: emscripten
  dev_url: emscripten
```

conda and wasm

- ☑ Add *emscripten32*:
  - ☑ Add *emscripten32* as plattform to { conda, conda-build }
  - ☑ Add compiler package for *emscripten32*
- ☐ Build simple packages: bzip2, sqlite, ...

conda and wasm

- ☑ Add *emscripten32*:
  - ☑ Add *emscripten32* as plattform to { conda, conda-build }
  - ☑ Add compiler package for *emscripten32*
- ☑ Build simple packages: bzip2, sqlite, ...

CONDA conda and wasm

- ☑ Add *emscripten32*:
  - ☑ Add *emscripten32* as plattform to { conda, conda-build }
  - ☑ Add compiler package for *emscripten32*
- ☑ Build simple packages: bzip2, sqlite, ...
- ☐ Build Python

conda and wasm

python package – recipe.yaml

```yaml
context:
  version: 3.11.0
package:
  name: python
  version: '{{ version }}'
source:
  git_url: https://github.com/python/cpython
  git_rev: 12a2e41e8a276ec3a68f3e5f94b02752185c66fb
  patches:
    - sel(emscripten): patches/ctypes-dont-deref-function-pointer.patch
build:
  number: 0
requirements:
  build:
    - sel(unix):          make
    - sel(emscripten):    python
  host:
    - sqlite
    - libffi
    - zlib
```

conda and wasm

python package – build.sh

```bash
#!/usr/bin/env bash

if [[ $target_platform == "emscripten-32" ]]; then
    CONFIG_SITE=../../conda.config.site-wasm32-emscripten READELF=true \
    emconfigure ../../configure -C \
        CFLAGS="-fPIC -O2" \
        --host=wasm32-unknown-emscripten \
        --with-emscripten-target=browser \
        --with-build-python=python3.11 \
        -- ... \
        --prefix=${PREFIX}
    emmake make install
else
    # ...
fi
```

conda and wasm

- ☑ Add *emscripten32*:
  - ☑ Add *emscripten32* as plattform to { conda, conda-build }
  - ☑ Add compiler package for *emscripten32*
- ☑ Build simple packages: bzip2, sqlite, ...
- ☑ Build Python

conda and wasm

- ☑ Add *emscripten32*:
  - ☑ Add *emscripten32* as plattform to { conda, conda-build }
  - ☑ Add compiler package for *emscripten32*
- ☑ Build simple packages: bzip2, sqlite, ...
- ☑ Build Python
- ☐ Build Python Packages:

conda and wasm

- ☑ Add *emscripten32*:
  - ☑ Add *emscripten32* as plattform to { conda, conda-build }
  - ☑ Add compiler package for *emscripten32*
- ☑ Build simple packages: bzip2, sqlite, ...
- ☑ Build Python
- ☐ Build Python Packages:
  - ☐ CMake based
  - ☐ setup.py / setuptools based

conda and wasm

- ☑ Add *emscripten32*:
  - ☑ Add *emscripten32* as plattform to { conda, conda-build }
  - ☑ Add compiler package for *emscripten32*
- ☑ Build simple packages: bzip2, sqlite, ...
- ☑ Build Python
- ☐ Build Python Packages:
  - ☑ CMake based
  - ☐ setup.py / setuptools based

setuptools and cross-compiling

crossenv + cross-python = ❤

- *crossenv*[1]: Creates *vens* suitable for cross-compiling

[1] https://github.com/benfogle/crossenv
[2] https://anaconda.org/conda-forge/cross-python

setuptools and cross-compiling

crossenv + cross-python = ❤

- *crossenv*[1]: Creates *vens* suitable for cross-compiling
- conda *cross-python*[2]: Integrates *crossenv* into conda builds

[1] https://github.com/benfogle/crossenv
[2] https://anaconda.org/conda-forge/cross-python

setuptools and cross-compiling

crossenv + cross-python = ❤

- *crossenv*[1]: Creates *vens* suitable for cross-compiling
- conda *cross-python*[2]: Integrates *crossenv* into conda builds
- cross-compiling boils down to:

[1] https://github.com/benfogle/crossenv
[2] https://anaconda.org/conda-forge/cross-python

setuptools and cross-compiling

crossenv + cross-python = ❤

- *crossenv*[1]: Creates *vens* suitable for cross-compiling
- conda *cross-python*[2]: Integrates *crossenv* into conda builds
- cross-compiling boils down to:
  - Adding *crossenv* to the dependencies:

```yaml
# ...
requirements:
  build:
    - '{{ compiler("c") }}'
    - cross-python_emscripten32
```

[1] https://github.com/benfogle/crossenv
[2] https://anaconda.org/conda-forge/cross-python

setuptools and cross-compiling

crossenv + cross-python = ❤

- *crossenv*[1]: Creates *vens* suitable for cross-compiling
- conda *cross-python*[2]: Integrates *crossenv* into conda builds
- cross-compiling boils down to:
  - Adding *crossenv* to the dependencies:

```yaml
# ...
requirements:
  build:
    - '{{ compiler("c") }}'
    - cross-python_emscripten32
```

  - Trivial build step

```
python setup.py install
```

[1] https://github.com/benfogle/crossenv
[2] https://anaconda.org/conda-forge/cross-python

CONDA conda and wasm

- ☑ Add *emscripten32*:
  - ☑ Add *emscripten32* as plattform to { conda, conda-build }
  - ☑ Add compiler package for *emscripten32*
- ☑ Build simple packages: bzip2, sqlite, ...
- ☑ Build Python
- ☐ Build Python Packages:
  - ☑ CMake based
  - ☑ setuptools based

The Showcase

marring pybind11 and emscripten

emscripten + pybind11

= embind11

Emscripten

Transliterate JavaScript to C++

JavaScript:

```
var xhr = new XMLHttpRequest;
xhr.open("GET", "http://url");
```

Accessing JavaScript from C++:

```
val xhr = emscripten::val::global("XMLHttpRequest").new_();
xhr.call<void>("open", std::string("GET"), std::string("http://url"));
```

Emscripten

Transliterate JavaScript to C++

```javascript
// Get web audio api context
var AudioContext = window.AudioContext || window.webkitAudioContext;




// Got an AudioContext: Create context and OscillatorNode
var context = new AudioContext();
var oscillator = context.createOscillator();

// Configuring oscillator: set OscillatorNode type and frequency
oscillator.type = 'triangle';
oscillator.frequency.value = 261.63; // value in hertz - middle C

// Playing
oscillator.connect(context.destination);
oscillator.start();
```

Emscripten

Transliterate JavaScript to C++

```
// Get web audio api context
auto AudioContext = emscripten::val::global("AudioContext");
if (!AudioContext.as<bool>()) {
  AudioContext = emscripten::val::global("webkitAudioContext");
}

// Got an AudioContext: Create context and OscillatorNode
auto context = AudioContext.new_();
auto oscillator = context.call<emscripten::val>("createOscillator");

// Configuring oscillator: set OscillatorNode type and frequency
oscillator.set("type", emscripten::val("triangle"));
oscillator["frequency"].set("value", emscripten::val(261.63)); // Middle C

// Playing
oscillator.call<void>("connect", context["destination"]);
oscillator.call<void>("start", 0);
```

Make *em::val* available in python with pybind11:

```
py::class_<em::val>(m, "val",  py::dynamic_attr())
  .def_static( "get_global", [](std::string & arg){
    return em::val::global(arg.c_str());
  })
  .def("__getitem__",[](em::val * v, const std::string & key){
    return  v->operator[](key);
  })
  .def("__setitem__",[](em::val * v, std::string & key, em::val & val){
    return  v->set(key, val);
  })
  .def("call",[](em::val * v, const std::string & key, em::val & args){
    return v->call<em::val>(key.c_str(), args);
  })
;
```

Make *pybind11::object* available in JavaScript with 'emscripten::bind':

```
em::class_<py::object>("pyobject")

  .function("__call__",
    em::select_overload</*...*/>([](
      py::object & pyobject,
      em::val args
    ){
      return em::val(pyobject(args));
    }))
  ;
```

```python
import js

canvas = js.document.getElementById('mycanvas')
ctx = canvas.getContext("2d")
boundings = canvas.getBoundingClientRect()
top_left = boundings.left.pythonize(), boundings.top.pythonize()
is_down = [False]

def mouse_down(e):
    x = e.x.pythonize() - top_left[0]
    y = e.y.pythonize() - top_left[1]
    is_down[0] = True
    ctx.moveTo(x, y);

def mouse_move(e):
    if is_down[0]:
        x = e.x.pythonize() - top_left[0]
        y = e.y.pythonize() - top_left[1]
        ctx.lineTo(x, y);
        ctx.stroke();

def mouse_up(e):
    is_down[0] = False

canvas['onmousedown'] = js.js_callback(mouse_down)
canvas['onmousemove'] = js.js_callback(mouse_move)
canvas['onmouseup'] = js.js_callback(mouse_up)
```