Let’s embrace WebAssembly!

EuroPython 2018 - Edinburgh

Almar Klein

What is WebAssembly?

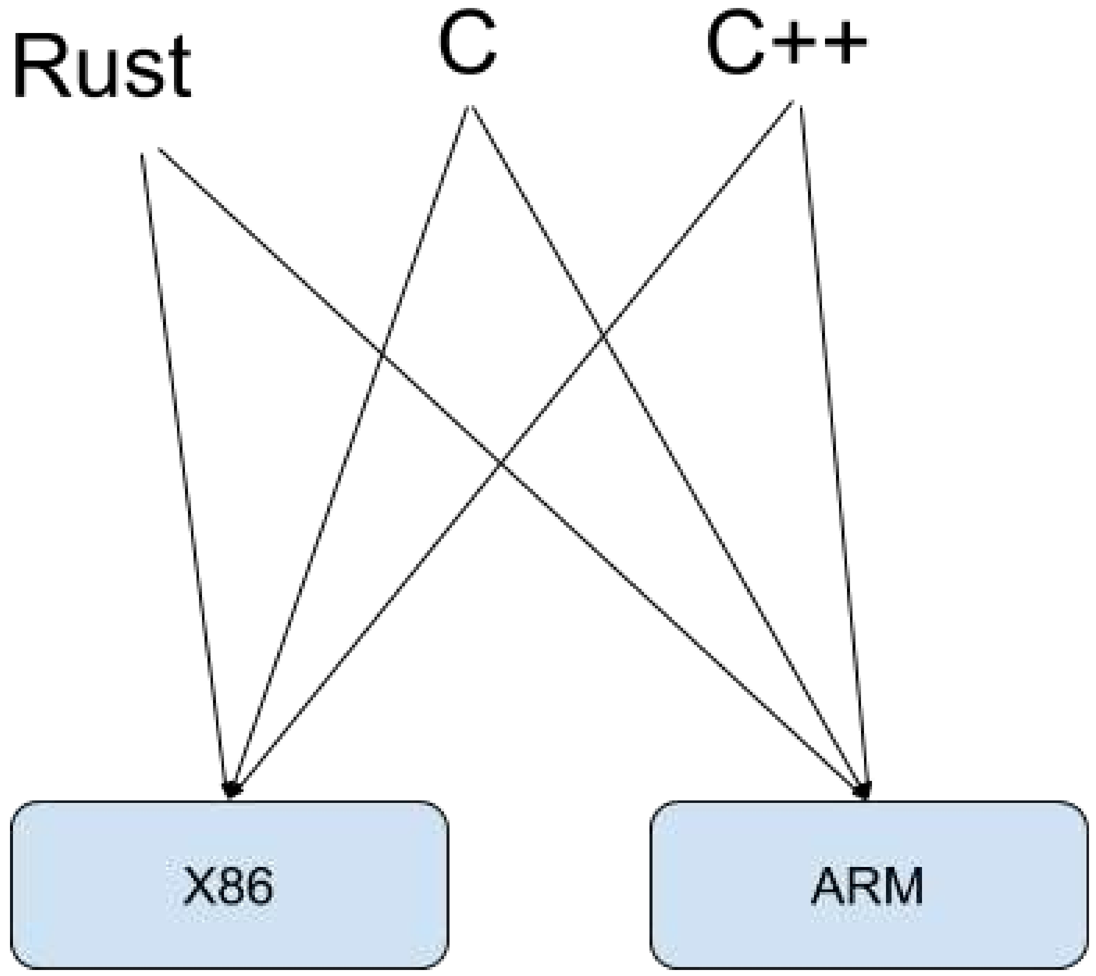
WebAssembly == WASM

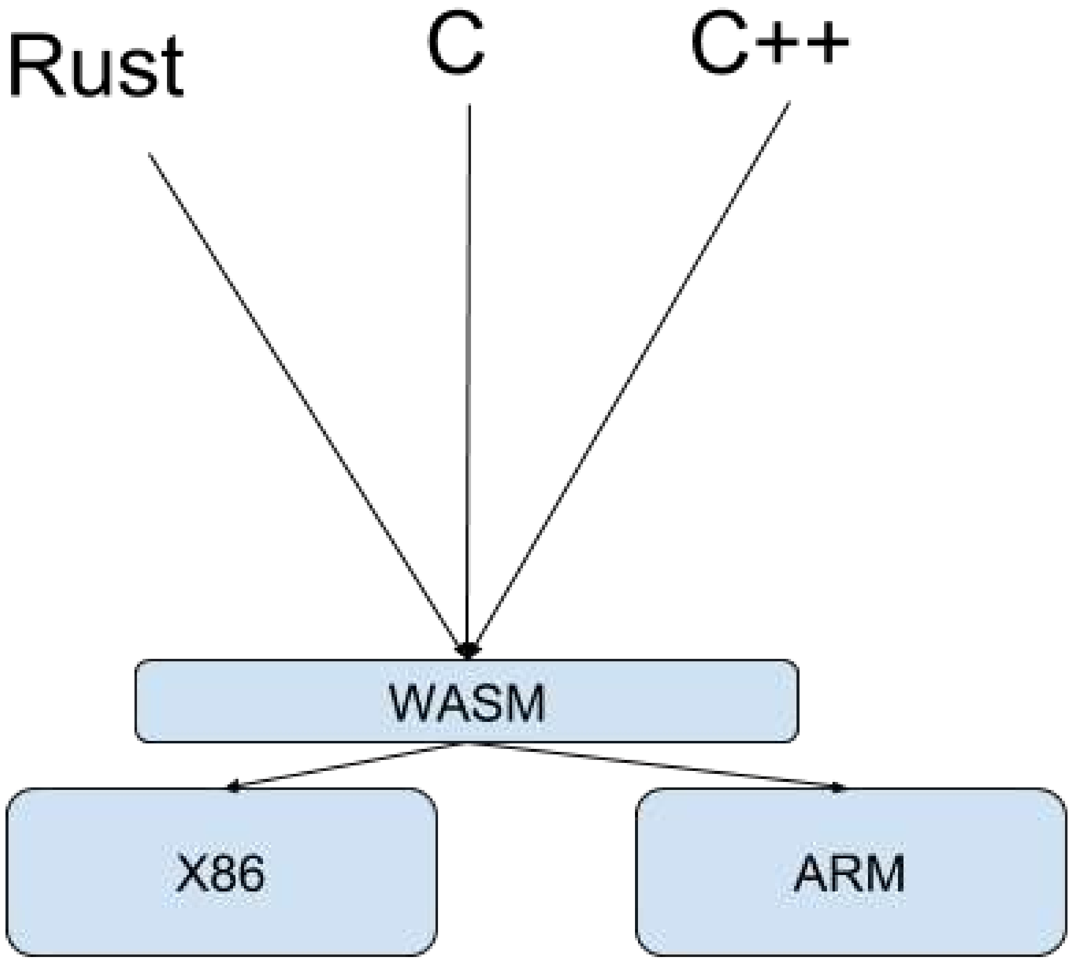
WASM is an OPEN standard ...

Collaborative effort by Mozilla, Google, Apple, Microsoft ...

… for executable code

It's fast!


Rust
C
C++
X86
ARM


Rust
C
C++
WASM
X86
ARM

It has a compact binary format

And a human readable counterpart:

```
wasm
(module
    (type $print (func (param i32))
    (func $main
        (i32.const 42)
        (call $print)
    )
    (start $main)
)
```

It’s safe

Because browsers.

WebAssembly is coming and it’s awesome!

SO MUCH
AWESOME
memegenerator.net

WebAssembly adoption

Lua community

Let's write web apps in Lua !!

Rust community

Let's use Rust for everything !!

C++ community

We can now write web apps in C++ ...

JavaScript community

Will this end our suffering?

Will this end our monopoly?

Python community

... what is this WebAssembly thing?

WASM may not be obvious for Python

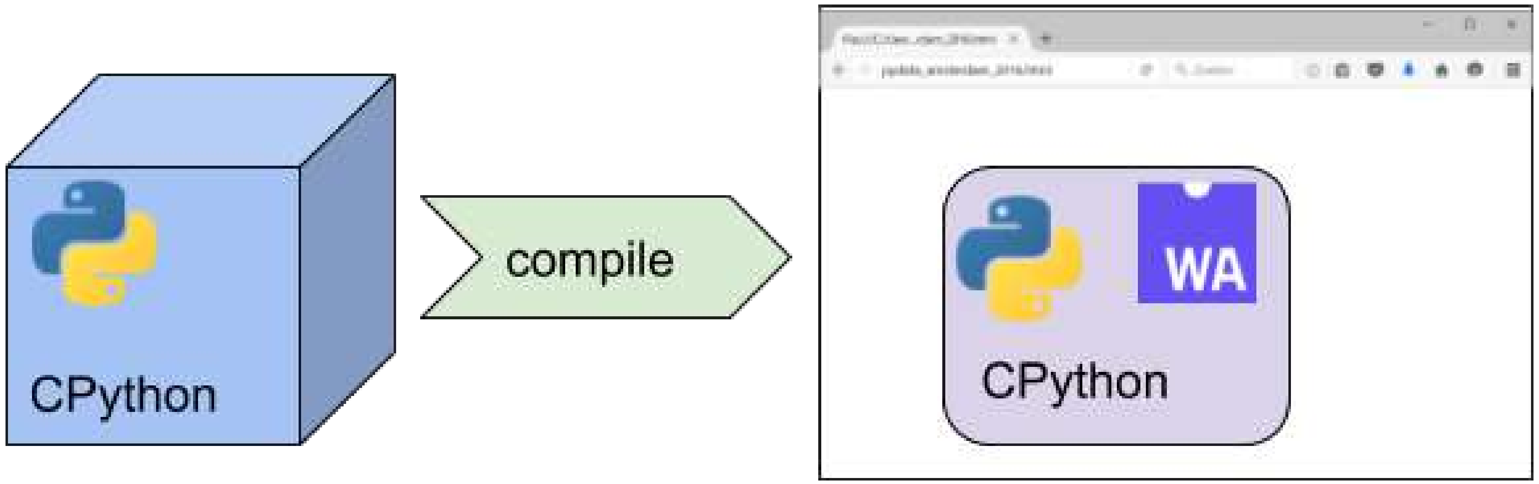
... Because Python is an *interpreted* language

Three use-cases how we can embrace WASM

In [ ]:

```
from ppci import wasm
```

Use case 1: Compile a Python interpreter

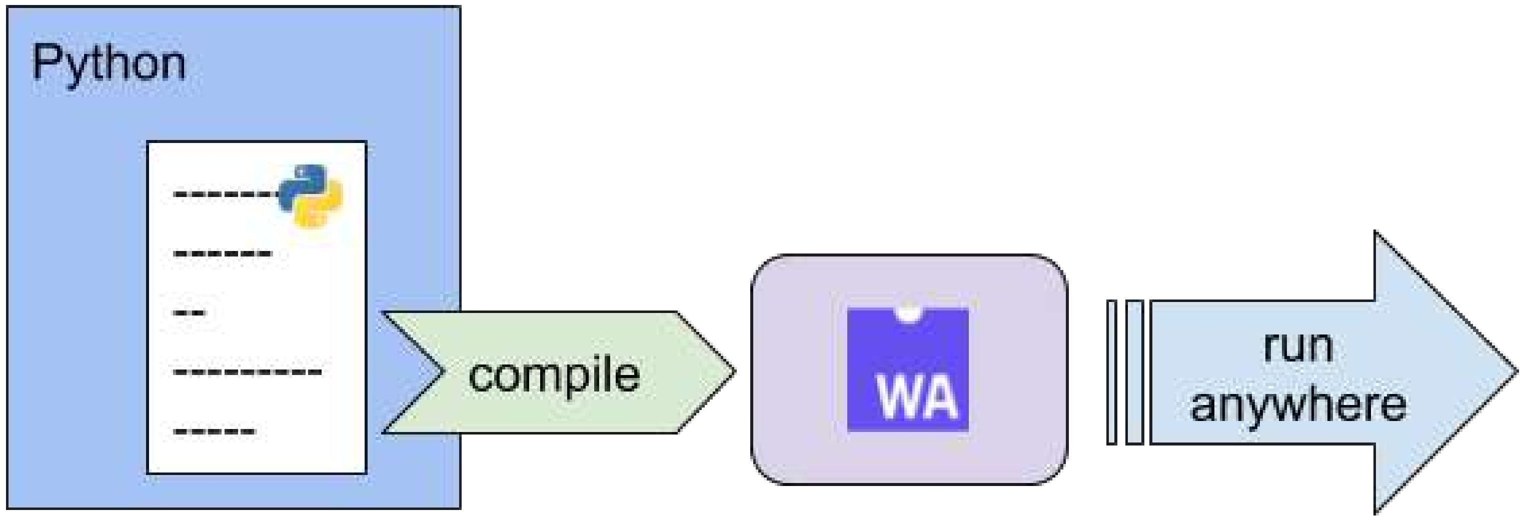

CPython
compile
WA
CPython

Examples

- Pyodide: compiles CPython + numpy/pandas/matplotlib, to run in the browser
- PyPyJS
- RustPython: Python interpreter written in Rust

Note: Python code is still run in a VM!

Use case 2: Compile a subset of Python to WASM


Python
compile
WA
run anywhere

```
In [ ]: @wasm.wasmify
        def find_prime(nth):
            n = 0
            i = -1
            while n < nth:
                i = i + 1
                if i <= 1:
                    continue  # nope
                elif i == 2:
                    n = n + 1
                else:
                    gotit = 1
                    for j in range(2,  i//2+1):
                        if i % j == 0:
                            gotit = 0
                            break
                    if gotit == 1:
                        n = n + 1
            return i
```

```
In [ ]: %time find_prime(1000)
```

Run in JS

In [ ]:
```
from ppci.lang.python import python_to_wasm

def main():
    print(find_prime(1000))

m = python_to_wasm(main, find_prime)
```

In [ ]:
```
wasm.run_wasm_in_notebook(m)
```

Compile a subset of Python to WASM

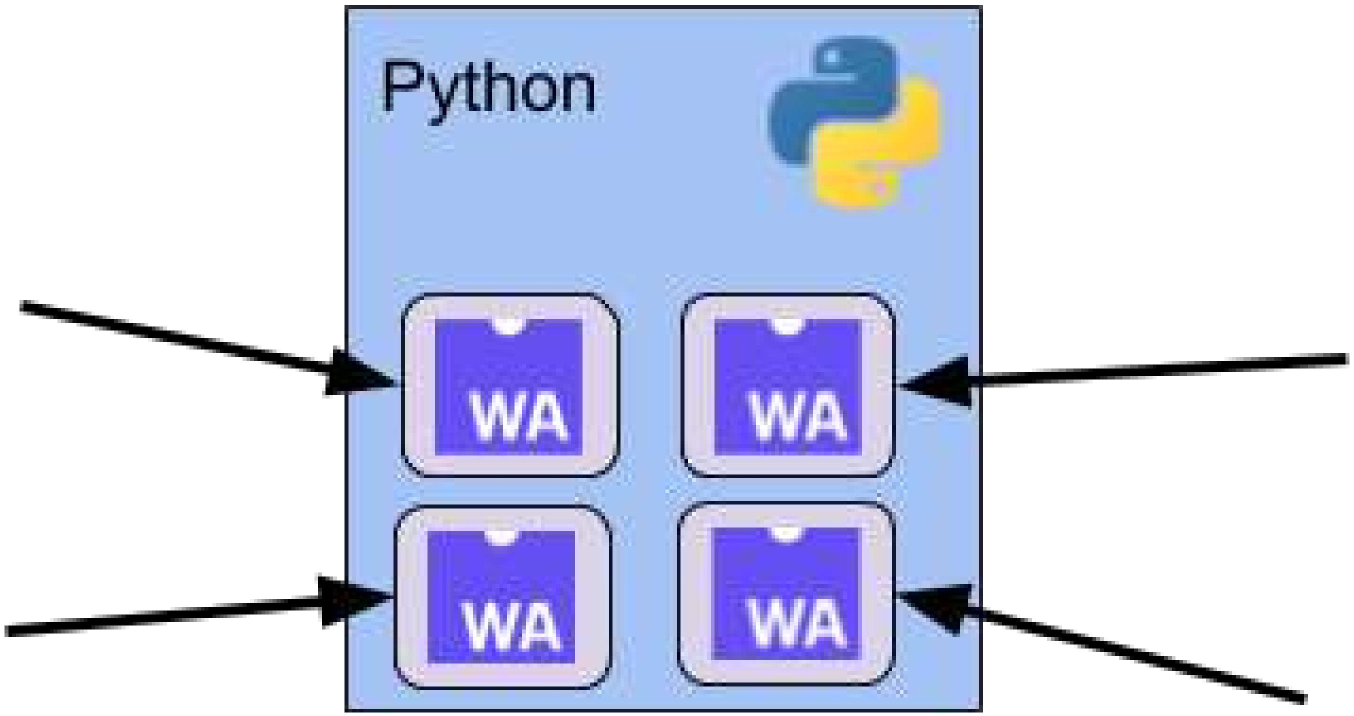
- Write code to run on the web
- Write code to run fast
- Binaries are cross-platform!

Note:

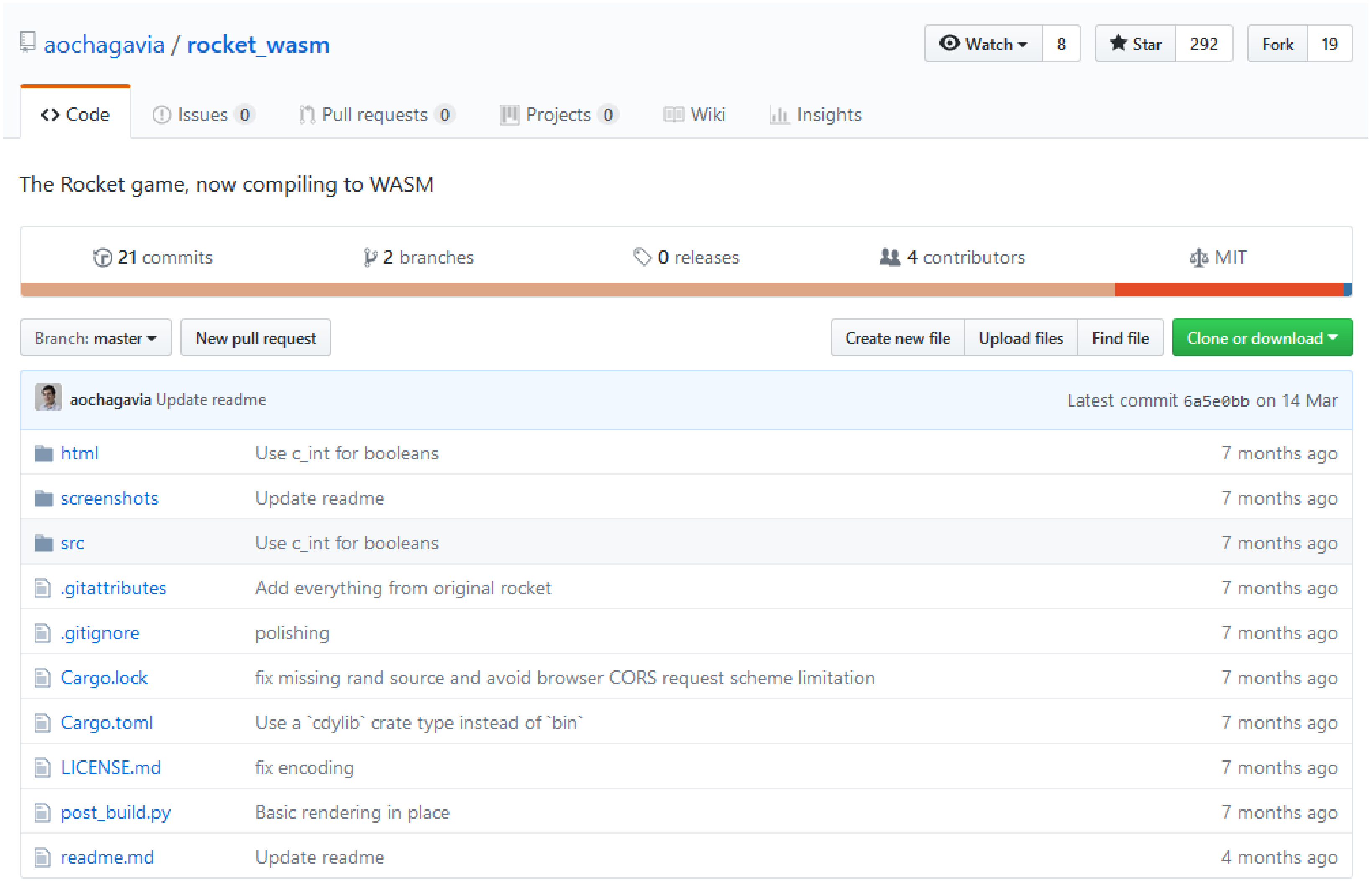
- The python-to-wasm compiler is just a POC!
- Assumes a (reliable) wasm-to-native compiler

Use case 3: Python as a platform to bind and run WASM modules

... and allow that code to call into Python functions


Python
WA
WA
WA
WA

Rocket game



aochagavia / rocket_wasm

The Rocket game, now compiling to WASM

21 commits | 2 branches | 0 releases | 4 contributors | MIT

aochagavia Update readme — Latest commit 6a5e0bb on 14 Mar

html	Use c_int for booleans	7 months ago
screenshots	Update readme	7 months ago
src	Use c_int for booleans	7 months ago
.gitattributes	Add everything from original rocket	7 months ago
.gitignore	polishing	7 months ago
Cargo.lock	fix missing rand source and avoid browser CORS request scheme limitation	7 months ago
Cargo.toml	Use a `cdylib` crate type instead of `bin`	7 months ago
LICENSE.md	fix encoding	7 months ago
post_build.py	Basic rendering in place	7 months ago
readme.md	Update readme	4 months ago

(rocket.html)

Single binary WASM file (58 KB)

aochagavia Use c_int for booleans

..

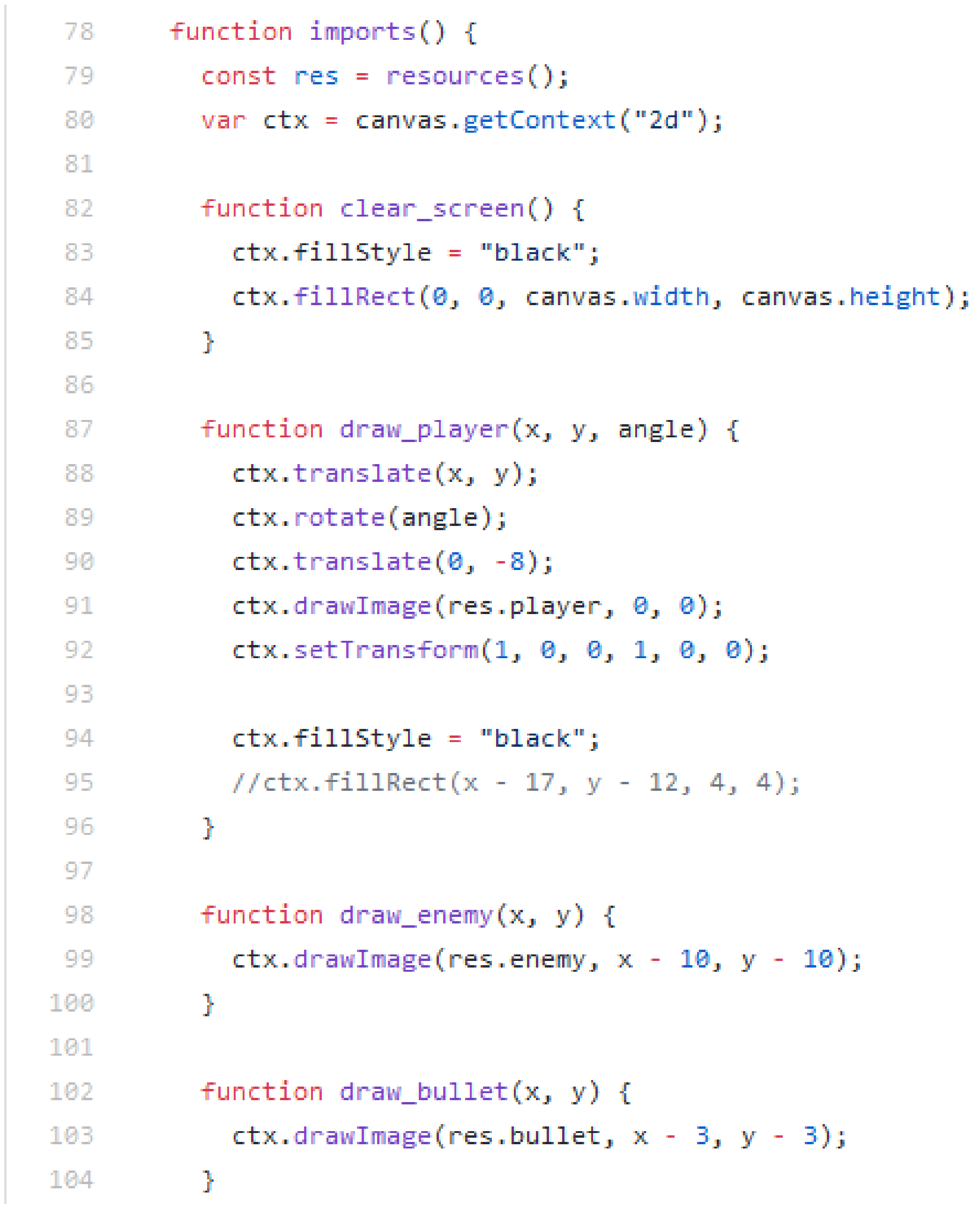
index.html Use c_int for booleans

program.wasm Use c_int for booleans

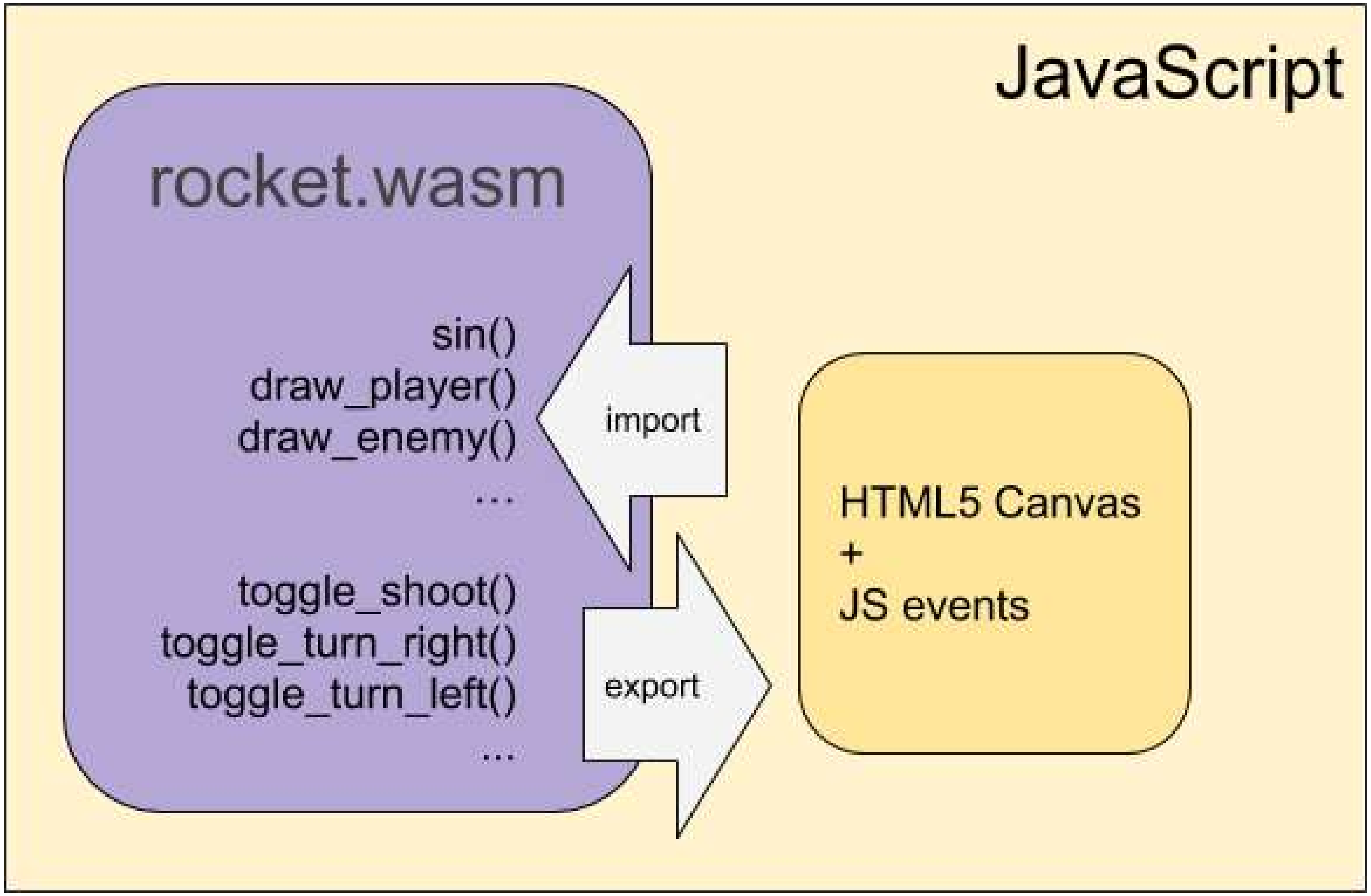
In [ ]:

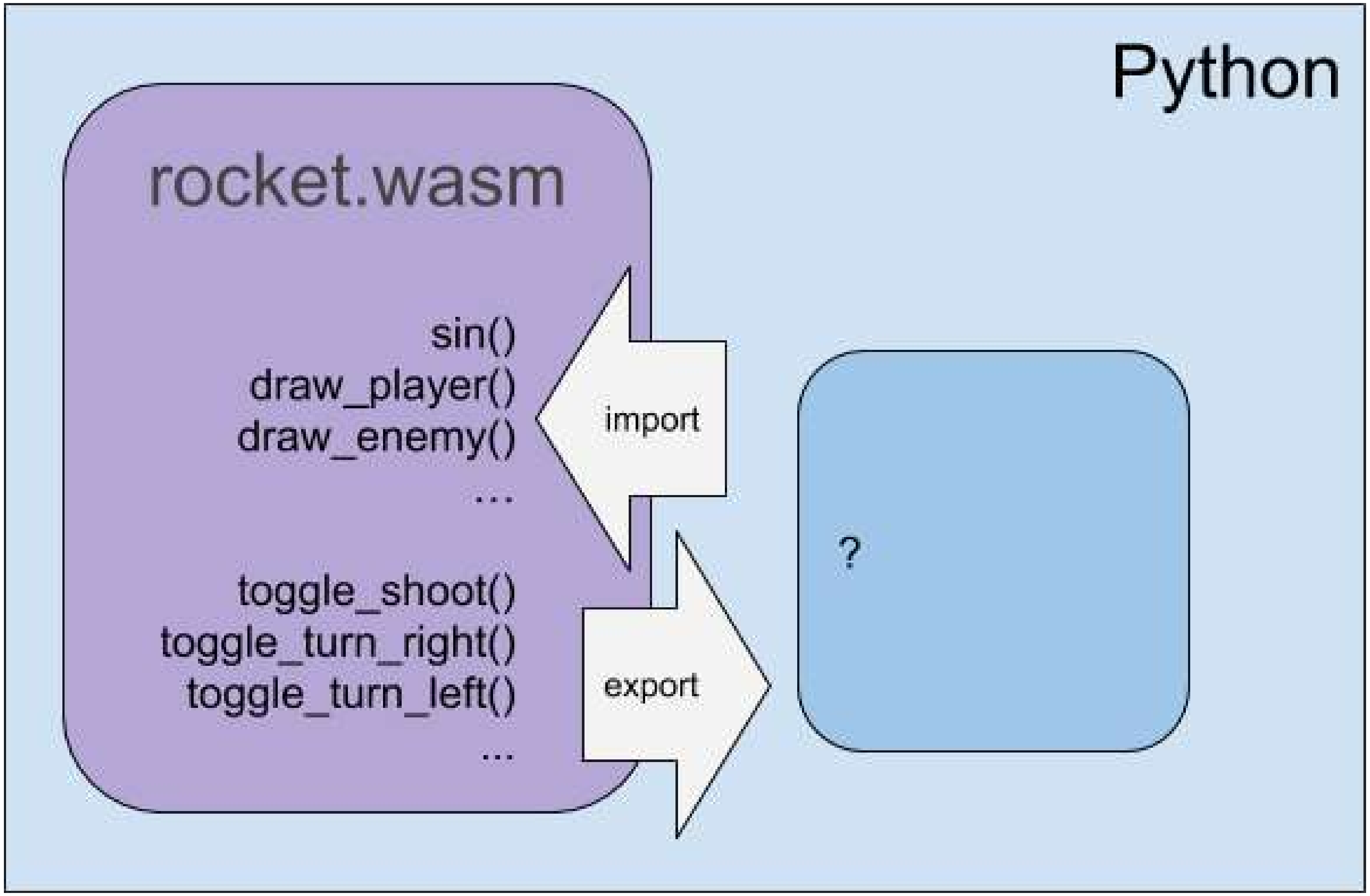
```
from ppci import wasm

m = wasm.Module(open(r'wasm/rocket.wasm', 'rb'))
m
```

In [ ]:
```
m.show_interface()
```



```
function imports() {
  const res = resources();
  var ctx = canvas.getContext("2d");

  function clear_screen() {
    ctx.fillStyle = "black";
    ctx.fillRect(0, 0, canvas.width, canvas.height);
  }

  function draw_player(x, y, angle) {
    ctx.translate(x, y);
    ctx.rotate(angle);
    ctx.translate(0, -8);
    ctx.drawImage(res.player, 0, 0);
    ctx.setTransform(1, 0, 0, 1, 0, 0);

    ctx.fillStyle = "black";
    //ctx.fillRect(x - 17, y - 12, 4, 4);
  }

  function draw_enemy(x, y) {
    ctx.drawImage(res.enemy, x - 10, y - 10);
  }

  function draw_bullet(x, y) {
    ctx.drawImage(res.bullet, x - 3, y - 3);
  }
```


JavaScript
rocket.wasm
sin()
draw_player()
draw_enemy()
...
import
toggle_shoot()
toggle_turn_right()
toggle_turn_left()
...
export
HTML5 Canvas
+
JS events


Python
rocket.wasm
sin()
draw_player()
draw_enemy()
...
toggle_shoot()
toggle_turn_right()
toggle_turn_left()
...
import
export
?

In [ ]:

```python
class PythonRocketGame:

    # ...

    def wasm_sin(self, a:float) -> float:
        return math.sin(a)

    def wasm_cos(self, a:float) -> float:
        return math.cos(a)

    def wasm_Math_atan(self, a:float) -> float:
        return math.atan(a)

    def wasm_clear_screen(self) -> None:
        # ...

    def wasm_draw_bullet(self, x:float, y:float) -> None:
        # ...

    def wasm_draw_enemy(self, x:float, y:float) -> None:
        # ...

    def wasm_draw_particle(self, x:float, y:float, a:float) -> None:
        # ...

    def wasm_draw_player(self, x:float, y:float, a:float) -> None:
       # ...

    def wasm_draw_score(self, score:float) -> None:
        # ...
```

Run Rocket in Python with Qt

```
In [ ]: from rocket_qt import QtRocketGame
        game = QtRocketGame()
```

```
In [ ]: game.run()
```

Run Rocket in Python with prompt_toolkit

Over SSH :)

This game is not that hard to play …

Let’s make an AI!

In [ ]:
```
#print(open('wasm/ai2.c', 'rt').read())
```

In [ ]:
```
from ppci import wasm
ai2 = wasm.Module(open('wasm/ai2.wasm', 'rb'))
```

In [ ]:
```
ai2.show_interface()
```

In [ ]:
```
from rocket_ai import AiRocketGame
game = AiRocketGame(ai2)
game.run()
```

Wrapping up ...

WASM is coming, and its awesome!

- Open, low-level, fast, compact and safe
- Already works in most browsers
- Not limited to the web

We Pythonista's should embrace it!

- E.g. run a Python VM in the browser
- E.g. compile subset of Python to fast, crossplatform code
- E.g. use Python as a platform to bind and execute WASM modules