WebAssembly

Web más alla de JS

¿Qué es WASM?

WASM...

- Es un formato de **código binario y portátil** (bytecode) para la ejecución en el lado del cliente.
- Es un **lenguaje de destino** de compilación de C, C++, Rust, Go, entre otros.
- Busca habilitar apps de **alto rendimiento en páginas web.**

C++ WASM ( emscripten)

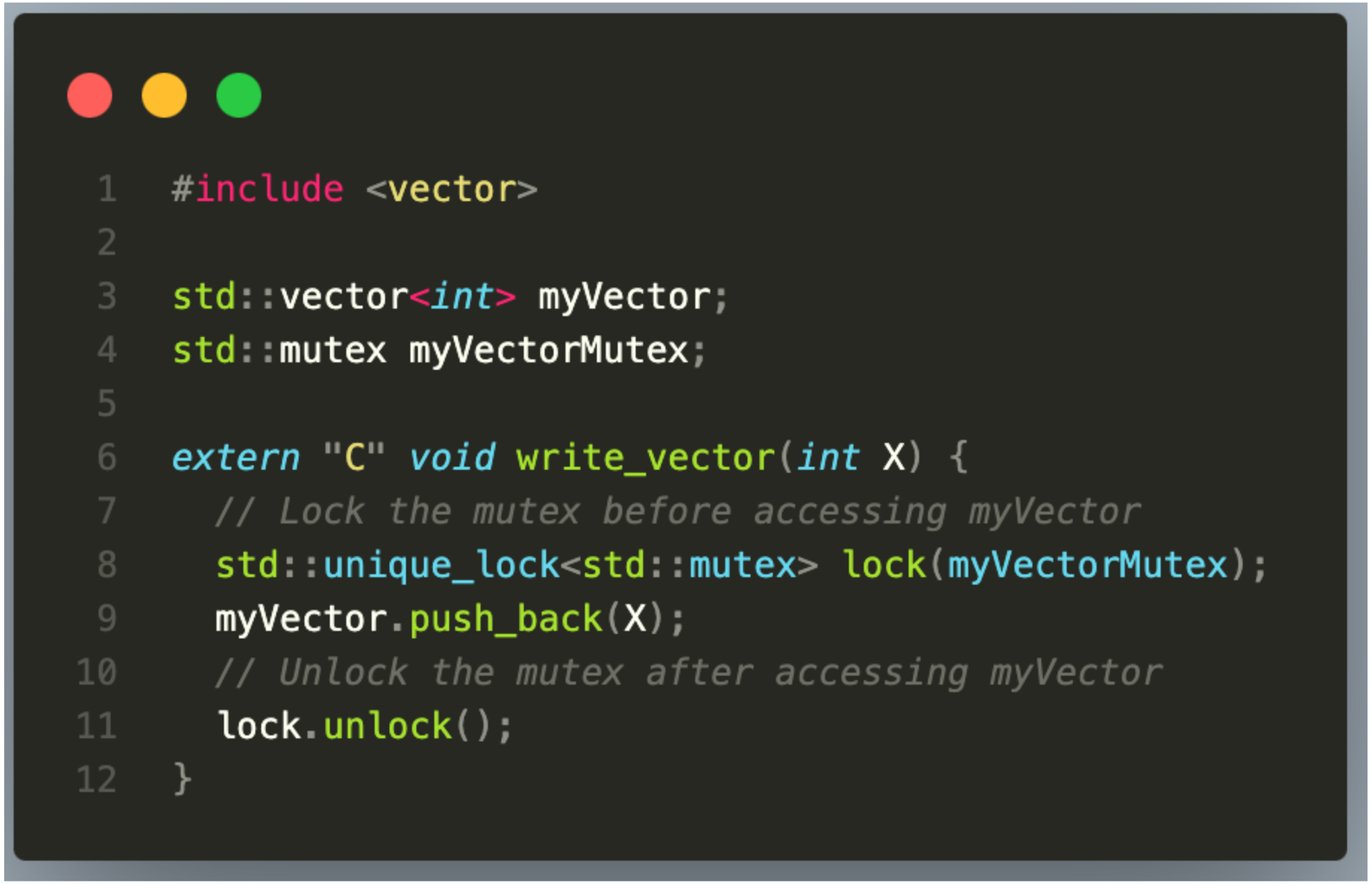
Emscripten es una herramienta completa de compilador OpenSource para WebAssembly.

Usando Emscripten, puedes: compilar código en C y C++, o cualquier otro lenguaje que utilice LLVM, en WebAssembly, y ejecutarlo en la web, Node.js u otras plataformas que admitan Wasm.

https://emscripten.org/docs/getting_started/downloads.html

```
emcc path_to_c_file.cpp -O3 -s
EXPORTED_FUNCTIONS=_main,_assignJobs,_write_vector,_malloc,_free -s
EXPORTED_RUNTIME_METHODS=ccall -s EXPORT_ES6=1 -o path_to_js_file.js
```

```
"cpp-emscripten": async () => {
  const { default: emscripten } = await import("../func/emscripten/scheduler.js");
  const { _assignJobs, _write_vector } = await emscripten();
  return (bins, durations) => {
    for (const duration of durations) _write_vector(duration);
    return _assignJobs(bins, durations.length);
  };
},
```



```cpp
#include <vector>

std::vector<int> myVector;
std::mutex myVectorMutex;

extern "C" void write_vector(int X) {
  // Lock the mutex before accessing myVector
  std::unique_lock<std::mutex> lock(myVectorMutex);
  myVector.push_back(X);
  // Unlock the mutex after accessing myVector
  lock.unlock();
}
```

```
const { m, times } = getParams();

const { default: emscripten } = await import("../func/emscripten/scheduler.js");
const { _assignJobs, _write_vector, _malloc, HEAPU32 } = await emscripten();
const timesArray = Uint32Array.from(times);
const timePtr = _malloc(timesArray.byteLength);
HEAPU32.set(timesArray, timePtr >> 2);

```

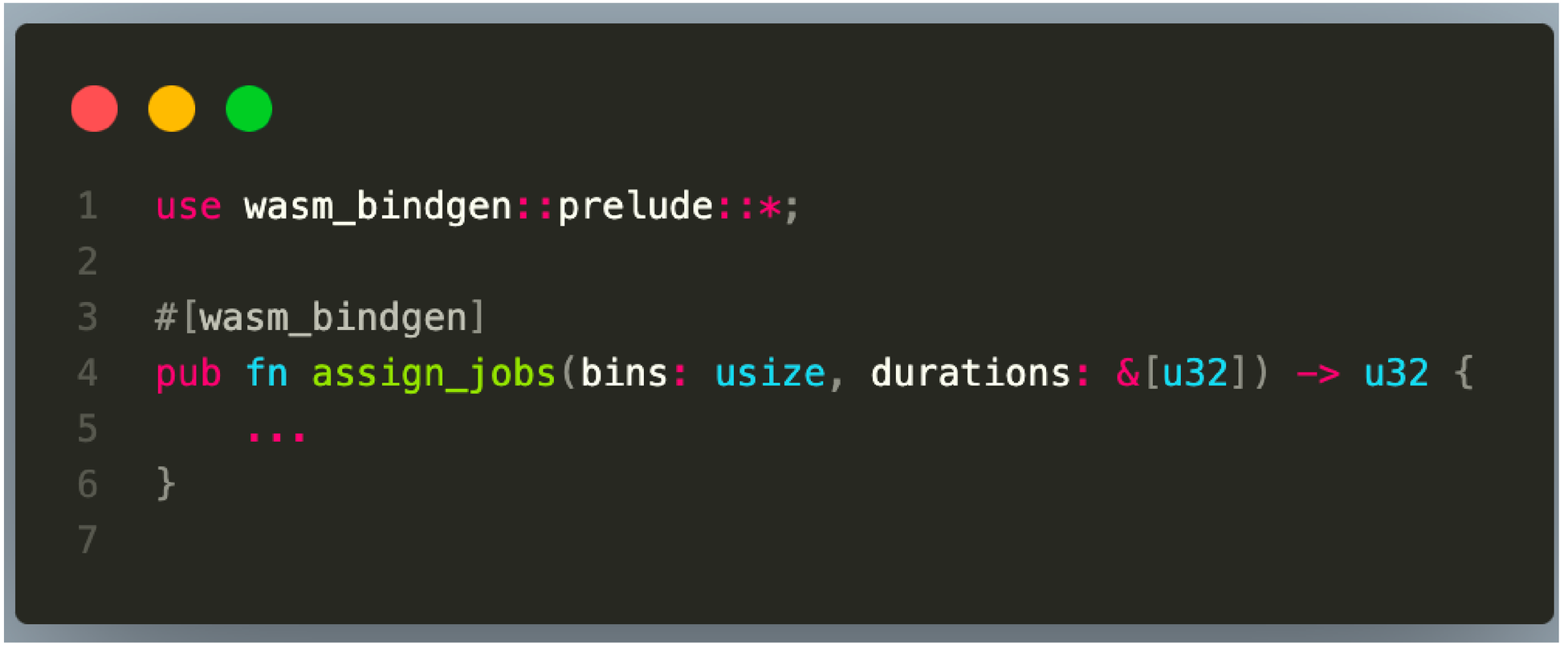
Rust WASM (wasm-pack)

Hay que instalar rust y rustsetup: https://rustup.rs/
Hay que instalar rust wasm: https://rustwasm.github.io/wasm-pack/installer/

Definir el Cargo.toml

```
wasm-pack build --release -m no-install --out-dir
path_to_rust_output_folder
```

```
1  "rust-wasm-pack": async () => {
2    const { assign_jobs } = await import("../func/wasm-pack/wasm-rust.js");
3    return (bins, durations) => assign_jobs(bins, durations);
4  },
```


use wasm_bindgen::prelude::*;
#[wasm_bindgen]
pub fn assign_jobs(bins: usize, durations: &[u32]) -> u32 {
...
}

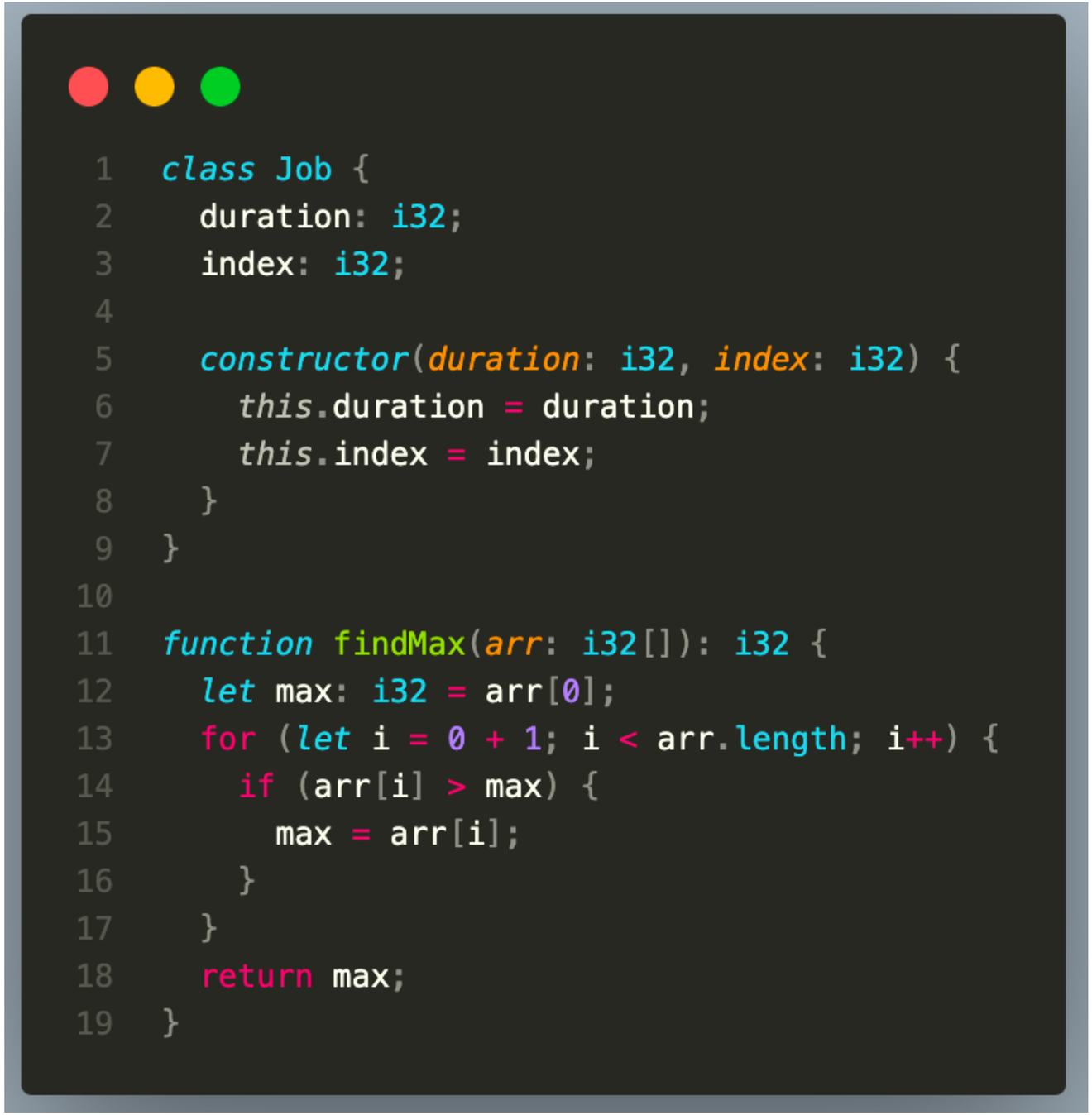
AssemblyScript WASM

Instalable con npm acá: https://www.npmjs.com/package/assemblyscript

Un TypeScript mucho más quisquilloso

```
asc path_to_assembly_script.ts--target release
```


assemblyscript: async () => {
const { assignJobs } = await import("../func/assemblyscript/dist/release");
return assignJobs;
},



```
class Job {
  duration: i32;
  index: i32;

  constructor(duration: i32, index: i32) {
    this.duration = duration;
    this.index = index;
  }
}

function findMax(arr: i32[]): i32 {
  let max: i32 = arr[0];
  for (let i = 0 + 1; i < arr.length; i++) {
    if (arr[i] > max) {
      max = arr[i];
    }
  }
  return max;
}
```

Demo

Conclusiones

	Velocidad	Facilidad de Uso	Fácil de Aprender	Librerías
JavaScript	x1.3 ~ x3	🤷‍♂️	🤷‍♂️	✅✅
C++ WASM	x1	❌	😬	✅✅
Rust WASM	x1*	✅	😵	✅
AssemblyScript WASM	x4*	✅	🤷‍♂️	😐
Go WASM	x8*	❌❌❌	😐	✅

Esto fue con los experimentos realizados.
Pueden existir errores.