DEVFEST
PARIS 2020

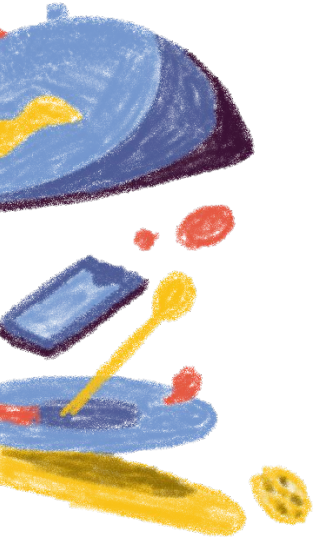
Hands on WebAssembly

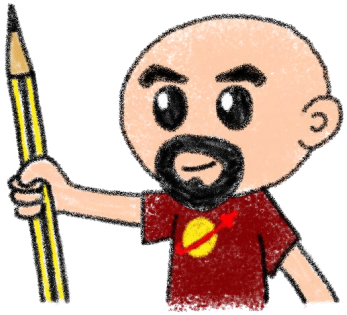
Horacio Gonzalez
@LostInBrittany

OVHcloud

Who are we?

Introducing myself and introducing ~~OVH~~ OVHcloud





Horacio Gonzalez

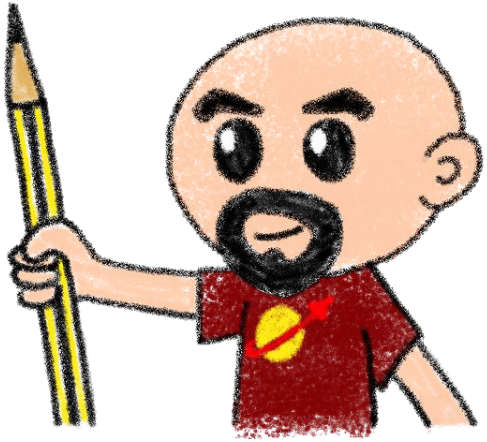
DEVFEST PARIS 2020

@LostInBrittany

Spaniard lost in Brittany, developer, dreamer and all-around geek

OVHcloud

DevRel Leader



DevFest du Bout du Monde

Finist Devs

Google Developers Experts 2019

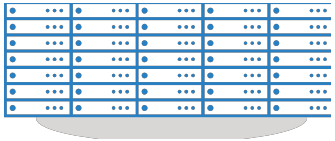
Web Technologies GDE

Flutter

OVHcloud: A Global Leader

200k Private cloud VMs running

1 **Dedicated IaaS Europe**



Hosting capacity : **1.3M** Physical Servers

360k Servers already deployed

Own **20Tbps** Netwok with **35 PoPs**

30 Datacenters

> 1.3M Customers in 138 Countries

OVHcloud: Our solutions

DEVFEST PARIS 2020

Cloud

- VPS
- Public Cloud
- Private Cloud
- Serveur dédié
- Cloud Desktop
- Hybrid Cloud

Mobile Hosting

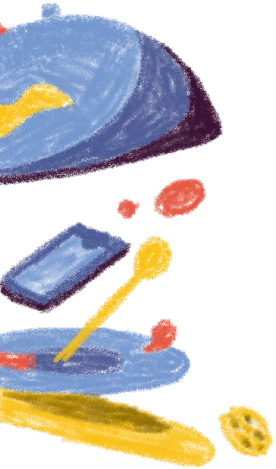
- Containers
- Compute
- Database
- Object Storage
- Securities
- Messaging

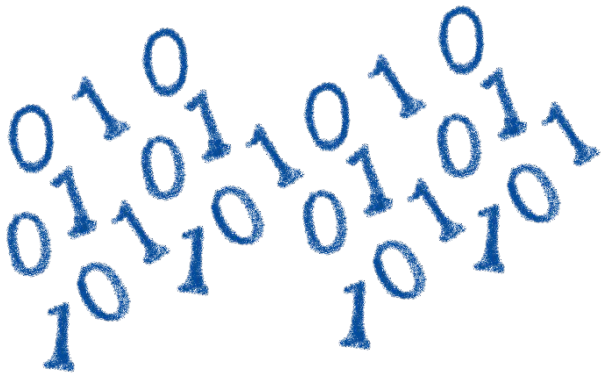
Web Hosting

- Domain names
- Email
- CDN
- Web hosting
- MS Office
- MS solutions

Telecom

- VoIP
- SMS/Fax
- Virtual desktop
- Cloud Storage
- Over the Box



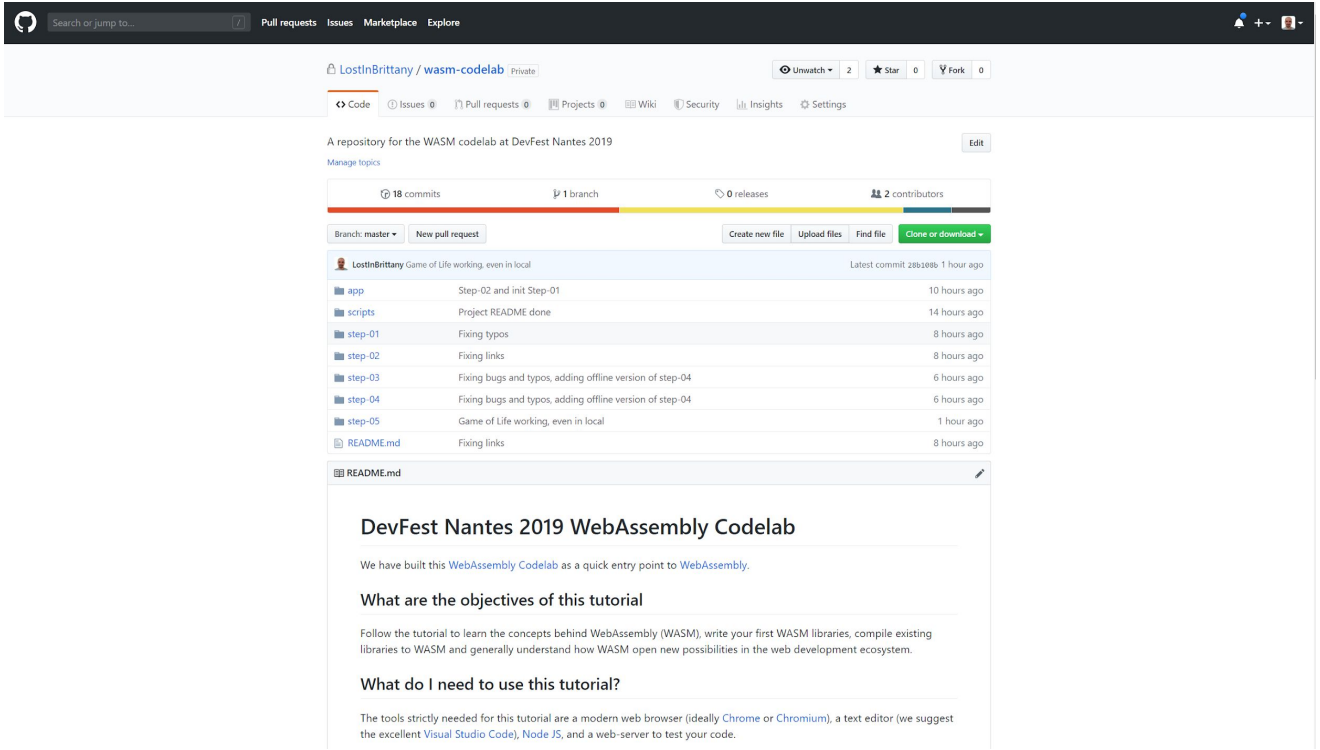


DEVFEST PARIS 2020

How is the codelab structured?

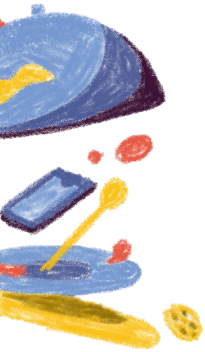
What are we coding today?

A GitHub repository



https://github.com/LostInBrittany/wasm-codelab

Nothing to install



C++11 -Os | COMPILE

```
int squarer(int num) {
  return num * num;
}
```

Wat | ASSEMBLE | DOWNLOAD

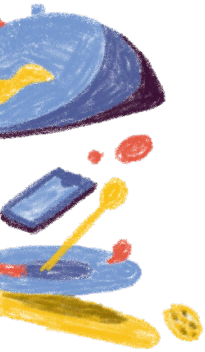
```
(module
  (type $type0 (func (param i32)
    (result i32)))
  (table 0 anyfunc)
  (memory 1)
  (export "memory" memory)
  (export "_Z7squareri" $func0)
  (func $func0 (param $var0 i32)
    (result i32)
    get_local $var0
    get_local $var0
    i32.mul
  )
)
```

Firefox x86 Assembly

```
wasm-function[0]:
  sub rsp, 8
  mov edx, edi
  mov ecx, edx
  mov eax, edx
  imul ecx, eax
  mov eax, ecx
  nop
  add rsp, 8
  ret
```

Using WebAssembly Explorer
and WebAssembly Studio

Only additional tool: a web server



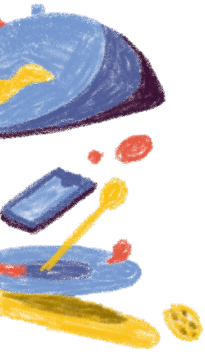
python™

node JS

NGINX

Because of the browser security model

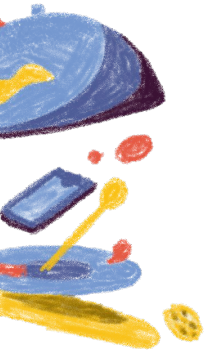
Procedure: follow the steps



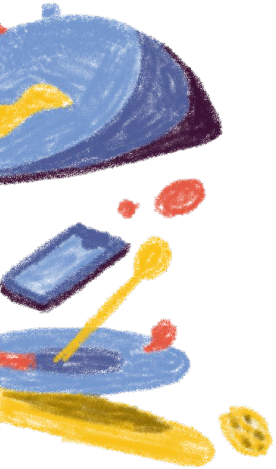
Step by step

But before coding, let's speak

DEVFEST PARIS 2020



What's this WebAssembly thing?



DEVFEST PARIS 2020

Did we say WebAssembly?

WASM for the friends...

WebAssembly, what's that?

What's WASM ?

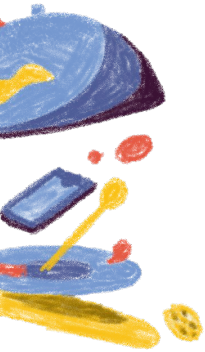
Does it replace JS ?

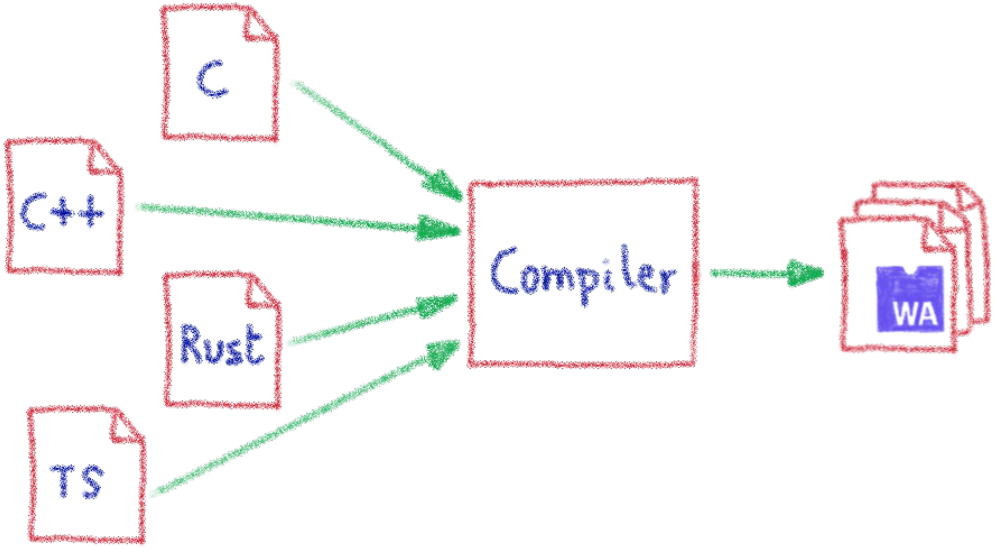
Can I code webapps in Rust?

Is HTML/CSS/JS stack obsolete?

Let's try to answer those (and other) questions...

A low-level binary format for the web

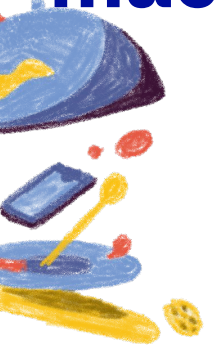


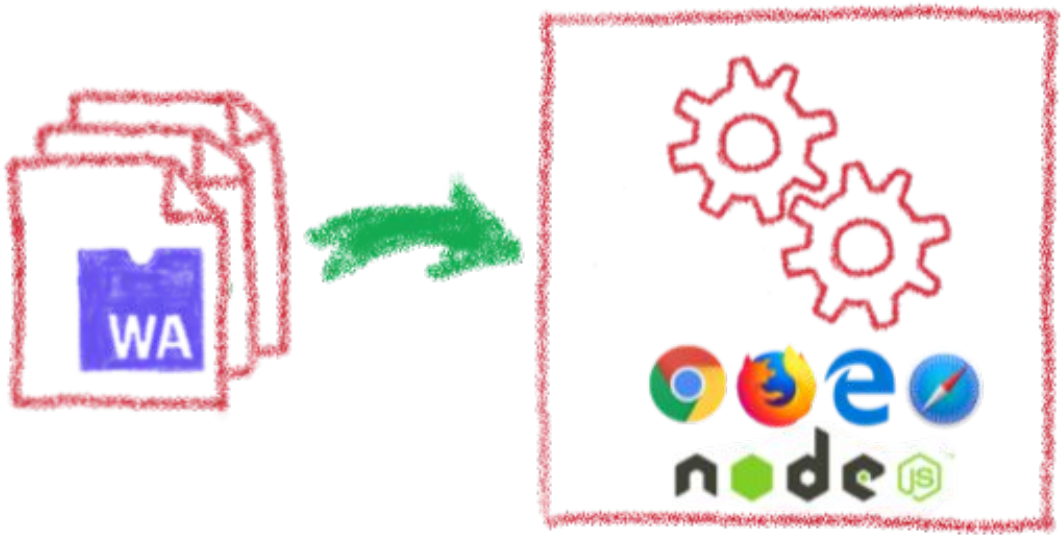


Not a programming language

A compilation target

That runs on a stack-based virtual machine





A portable binary format that runs on all modern browsers... but also on NodeJS!

With several key advantages

DEVFEST PARIS 2020

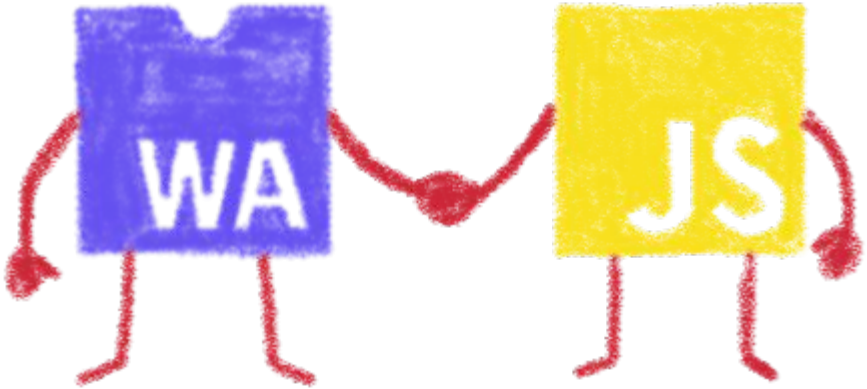
Fast & Efficient

Memory-safe & Sandboxed

Open & Deboggable

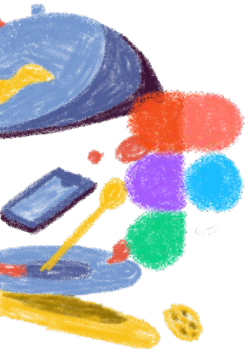
WWW Part of the Web Platform

But above all...



WebAssembly is not meant to replace JavaScript

Who is using WebAssembly today?



FIGMA

AUTOCAD

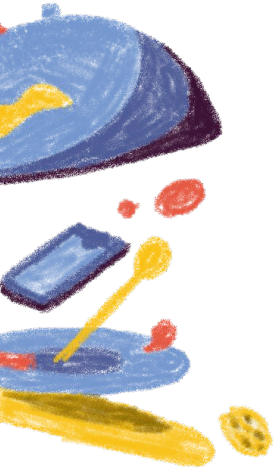
Qt

UNREAL ENGINE

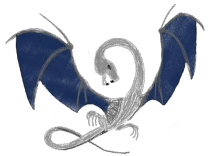
Google Earth

unity

And many more others...



Java Applets



DEVFEST PARIS 2020

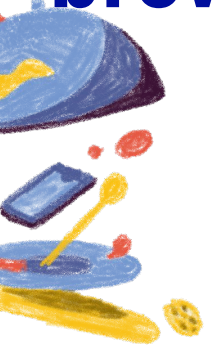
W3C

Macromedia Flash

A bit of history

Remembering the past
to better understand the present

Executing other languages in the browser

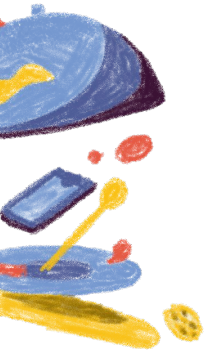


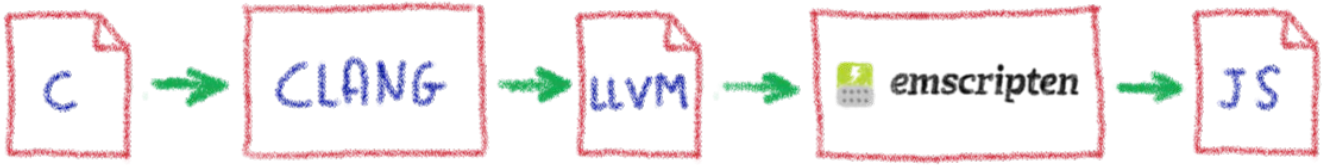
Java Applets

Macromedia Flash

A long story, with many failures...

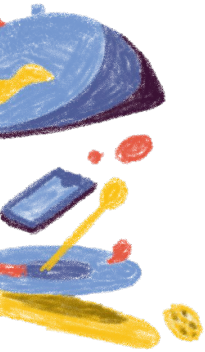
2012 - From C to JS: enter emscripten

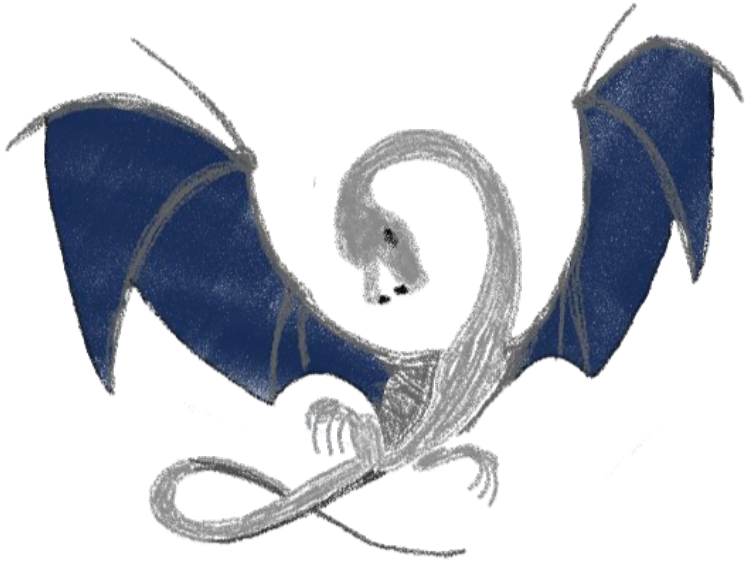




Passing by LLVM pivot

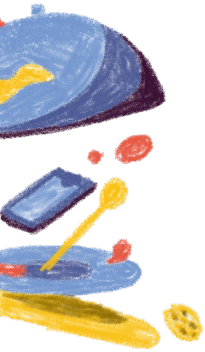
Wait, dude! What's LLVM?

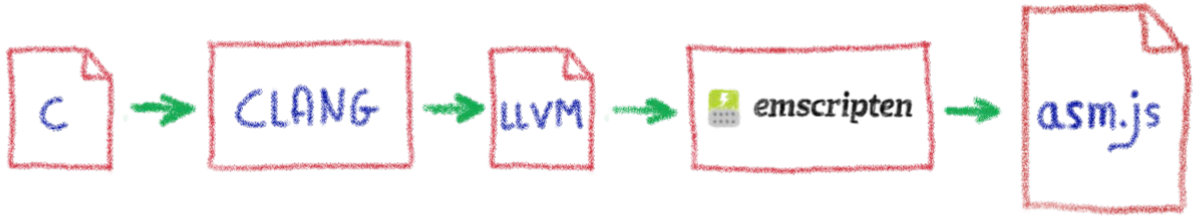




A set of compiler and toolchain technologies

2013 - Generated JS is slow...



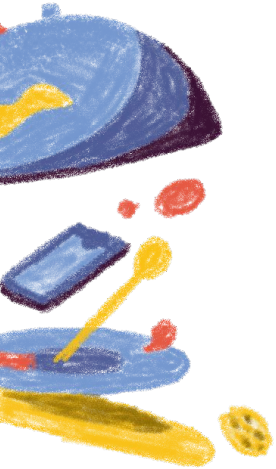


Let's use only a strict subset of JS: asm.js

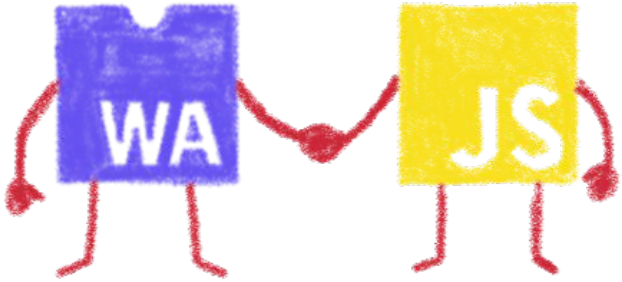
Only features adapted to AOT optimization

WebAssembly project

Joint effort



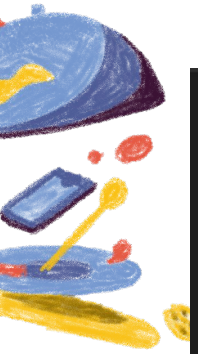
DEVFEST PARIS 2020



Hello W(ASM)orld

My first WebAssembly program

Do you remember your 101 C course?



```c
#include <stdio.h>

int main(int argc, char ** argv) {
  printf("Hello, world!\n");
}

```

A simple *HelloWorld* in C

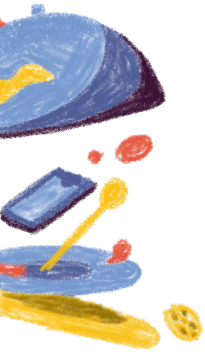
We compile it with emscripten

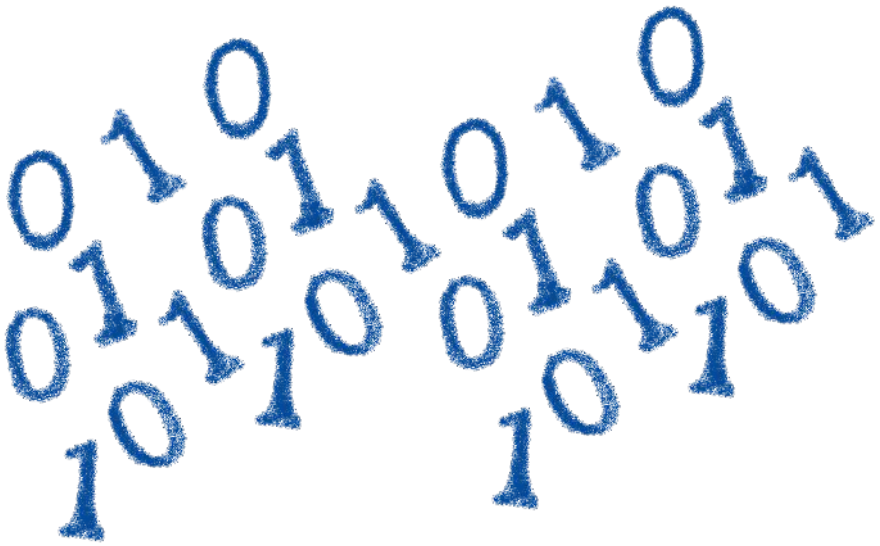
DEVFEST PARIS 2020

horacio@DESKTOP-6KHP1S2: ~/git/wasm/hello_world | horacio@DESKTOP-6KHP1S2: ~/git/emsdk

```
horacio@DESKTOP-6KHP1S2:~/git/wasm/hello_world$ emcc hello_world.c -o hello_world.html
cache:INFO: generating system asset: is_vanilla.txt... (this will be cached in "/home/horacio/.emscripten_cache/is_vanilla.txt" for subsequent builds)
cache:INFO:  - ok
shared:INFO: (Emscripten: Running sanity checks)
cache:INFO: generating system library: libcompiler_rt.bc... (this will be cached in "/home/horacio/.emscripten_cache/asmjs/libcompiler_rt.bc" for subsequent builds)
cache:INFO:  - ok
cache:INFO: generating system library: libc-wasm.bc... (this will be cached in "/home/horacio/.emscripten_cache/asmjs/libc-wasm.bc" for subsequent builds)
cache:INFO:  - ok
cache:INFO: generating system library: libdlmalloc.a... (this will be cached in "/home/horacio/.emscripten_cache/asmjs/libdlmalloc.a" for subsequent builds)
cache:INFO:  - ok
cache:INFO: generating system library: libpthreads_stub.bc... (this will be cached in "/home/horacio/.emscripten_cache/asmjs/libpthreads_stub.bc" for subsequent builds)
cache:INFO:  - ok
horacio@DESKTOP-6KHP1S2:~/git/wasm/hello_world$ ls
hello_world.c  hello_world.html  hello_world.js  hello_world.wasm
horacio@DESKTOP-6KHP1S2:~/git/wasm/hello_world$
```

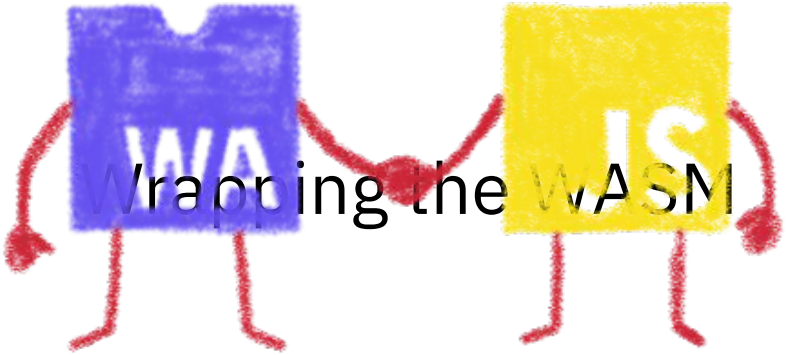
We get a .wasm file...



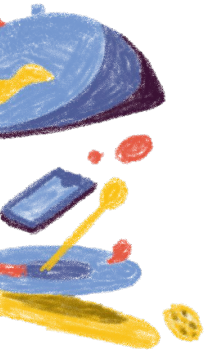


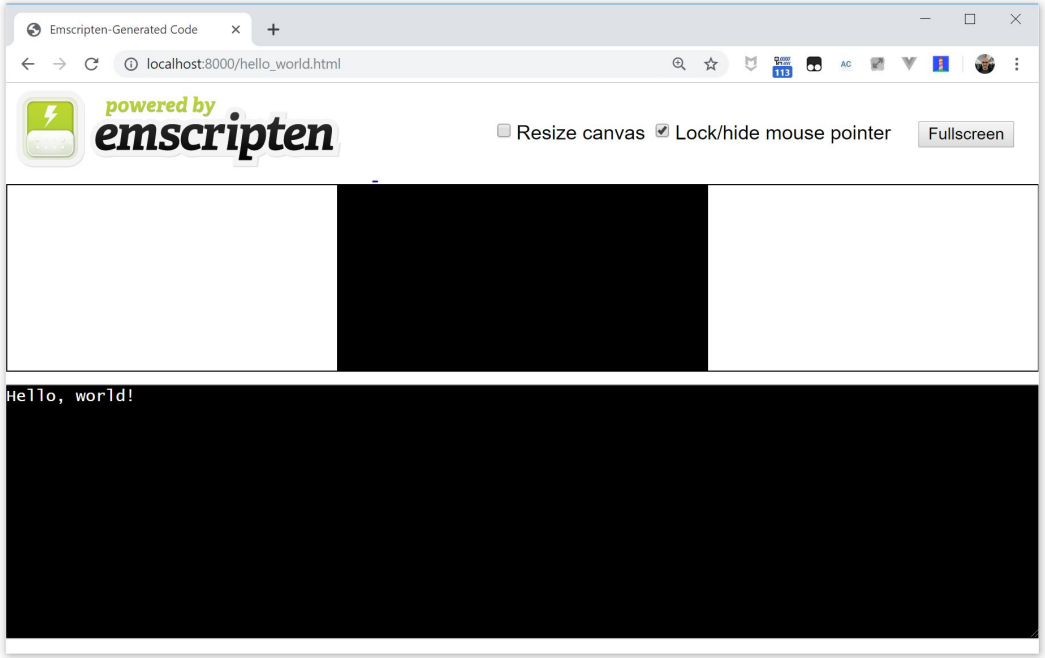
Binary file, in the binary WASM format

We also get a `.js` file...



And a .html file





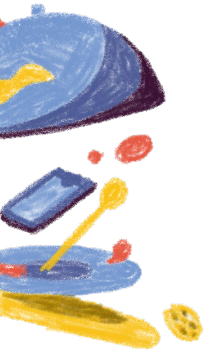
To quickly execute in the browser our WASM

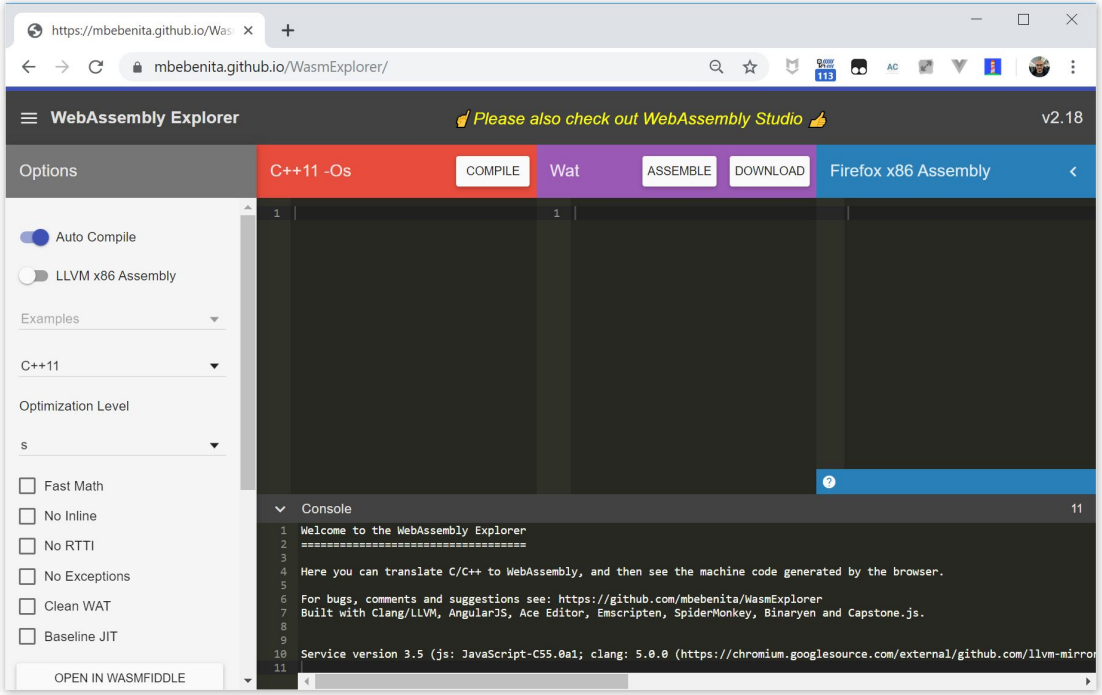
And in a more Real World™ case?

A simple process:

- Write or use existing code
  - In C, C++, Rust, Go, AssemblyScript...
- Compile
  - Get a binary `.wasm` file
- Include
  - The `.wasm` file into a project
- Instantiate
  - Async JavaScript compiling and instantiating the `.wasm` binary

I don't want to install a compiler now...

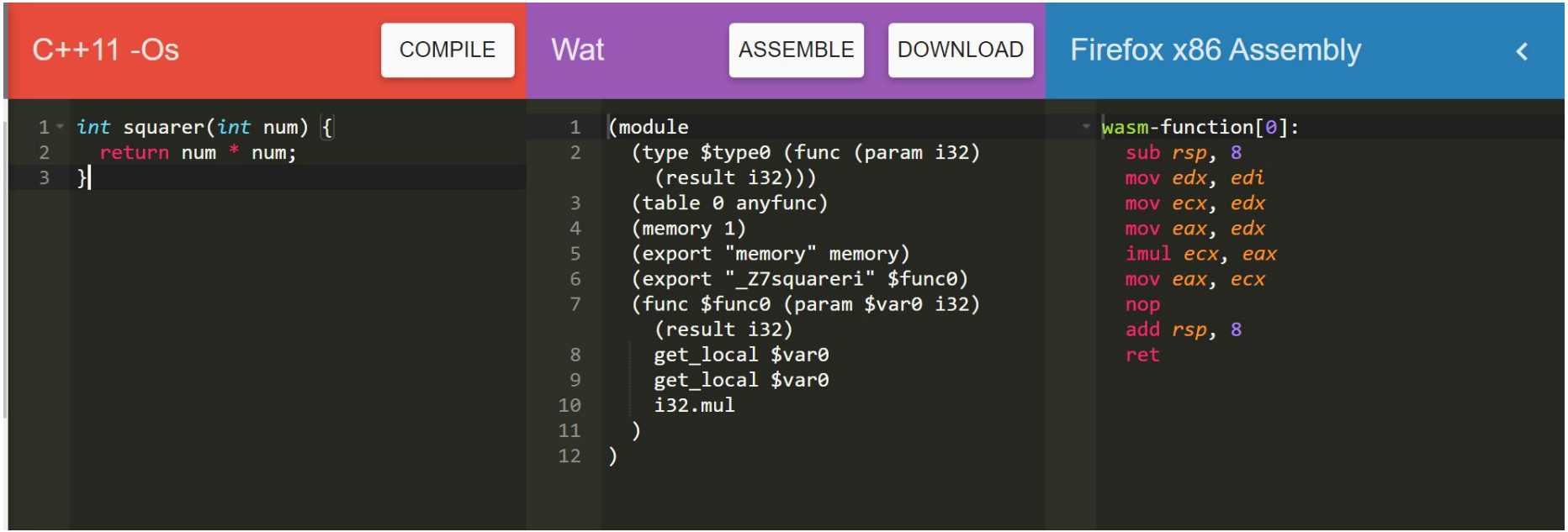




Let's use WASM Explorer

https://mbebenita.github.io/WasmExplorer/

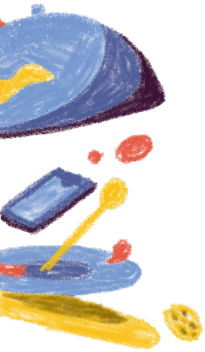
Let's begin with the a simple function

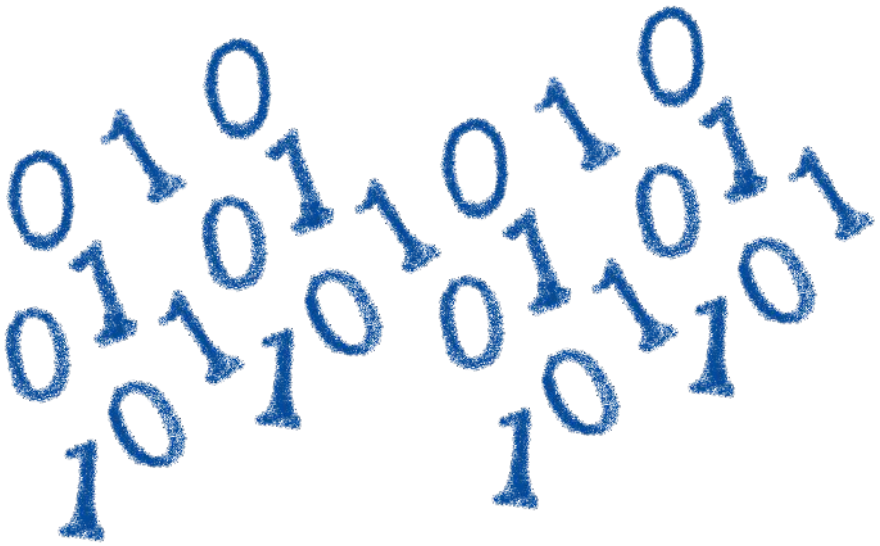


WAT: WebAssembly Text Format

Human readable version of the `.wasm` binary

Download the binary `.wasm` file





Now we need to call it from JS...

Instantiating the WASM

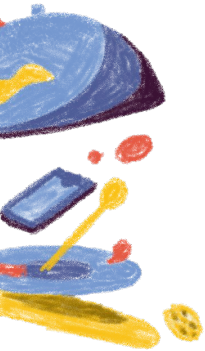
1. Get the `.wasm` binary file into an array buffer
2. Compile the bytes into a WebAssembly module
3. Instantiate the WebAssembly module

Instantiating the WASM

wasm > squarer > JS squarer.js > ...

```
3   var importObject = {
4       imports: {
5         imported_func: function(arg) {
6           console.log(arg);
7         }
8       }
9     };
10
11  async function loadWebAssembly() {
12      let response = await fetch('squarer.wasm');
13      let arrayBuffer = await response.arrayBuffer();
14      let wasmModule = await WebAssembly.instantiate(arrayBuffer, importObject);
15      squarer = await wasmModule.instance.exports._Z7squareri;
16      console.log('Finished compiling! Ready when you are...');
17  }
18
19  loadWebAssembly();
```

Loading the `squarer` function

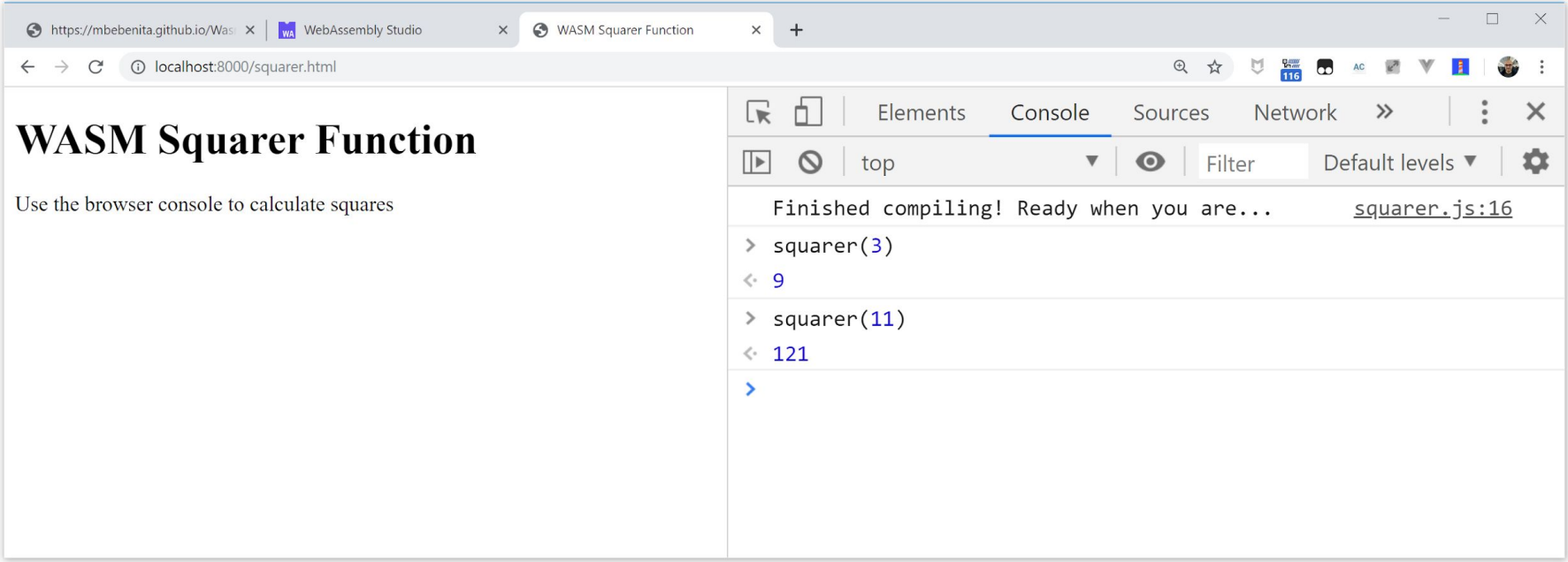


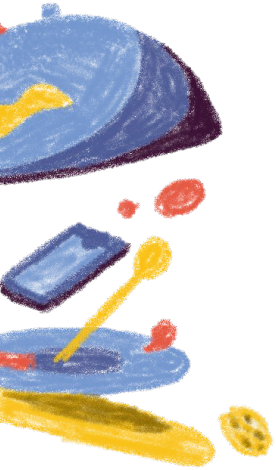
wasm > squarer > <> squarer.html > ...

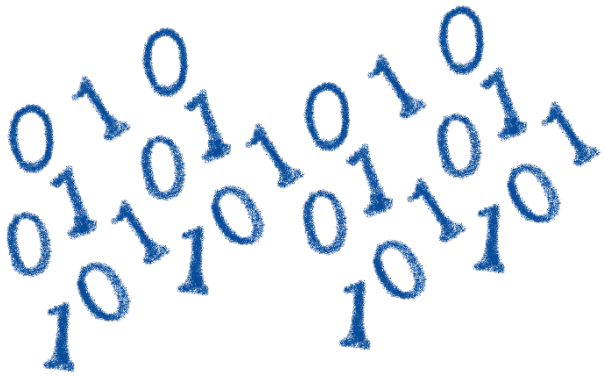
```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="utf-8" />
5      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6      <title>WASM Squarer Function</title>
7      <meta name="viewport" content="width=device-width, initial-scale=1">
8  </head>
9  <body>
10
11     <h1>WASM Squarer Function</h1>
12
13     <script src="squarer.js"></script>
14
15     <p>Use the browser console to calculate squares</p>
16 </body>
17 </html>
18
19
```

We instantiate the WASM by loading the wrapping JS

Using it!


https://mbebenita.github.io/Was
WebAssembly Studio
WASM Squarer Function
localhost:8000/squarer.html
WASM Squarer Function
Use the browser console to calculate squares
Elements
Console
Sources
Network
top
Filter
Default levels
Finished compiling! Ready when you are...
squarer.js:16
squarer(3)
9
squarer(11)
121



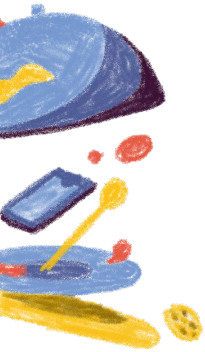


DEVFEST PARIS 2020

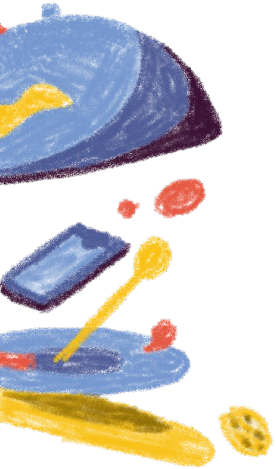
You sold us a codelab!

Stop speaking and let us code

You can do steps 01 and 02 now



Let's code, mates!



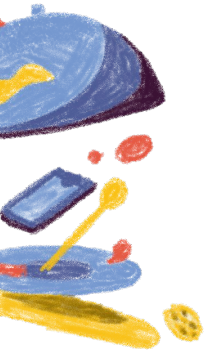
DEVFEST PARIS 2020

Wasmer

WASM outside the browser

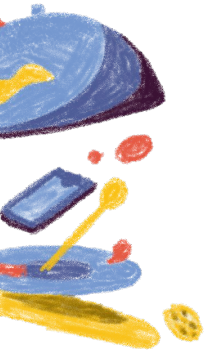
Not only for web developers

Run any code on any client... almost



Languages compiling to WASM

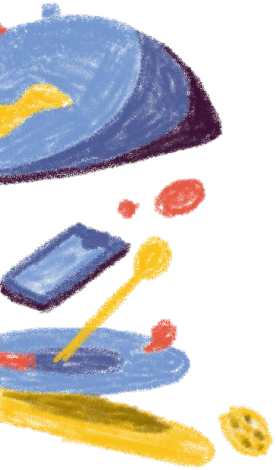
Includes WAPM



wapm install optipng

Oh, like npm for WASM!

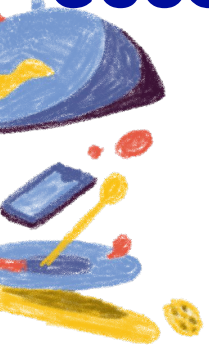
The WebAssembly Package Manager



Some use cases

What can I do with it?

Tapping into other languages ecosystems

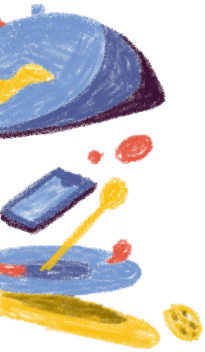


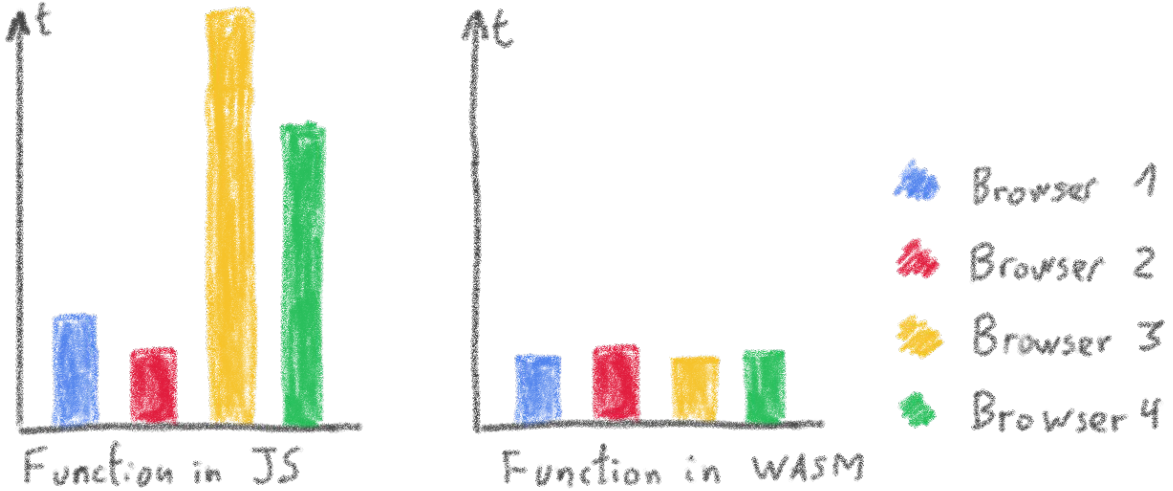
SQUOOSH.APP

OptiPNG (C)
Resize (Rust)
MozJPEG (C++)
webp (C)

Don't rewrite libs anymore

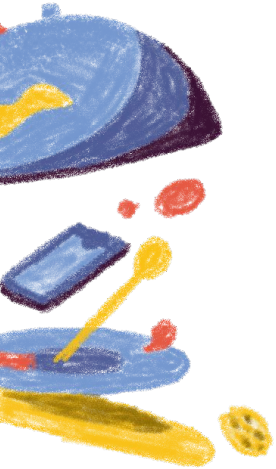
Replacing problematic JS bits





Predictable performance

Same peak performance, but less variation



DEVFEST PARIS 2020

Communicating between JS and WASM

Shared memory, functions...

Native WASM types are limited

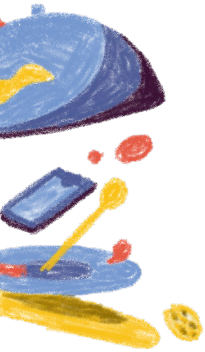
DEVFEST PARIS 2020

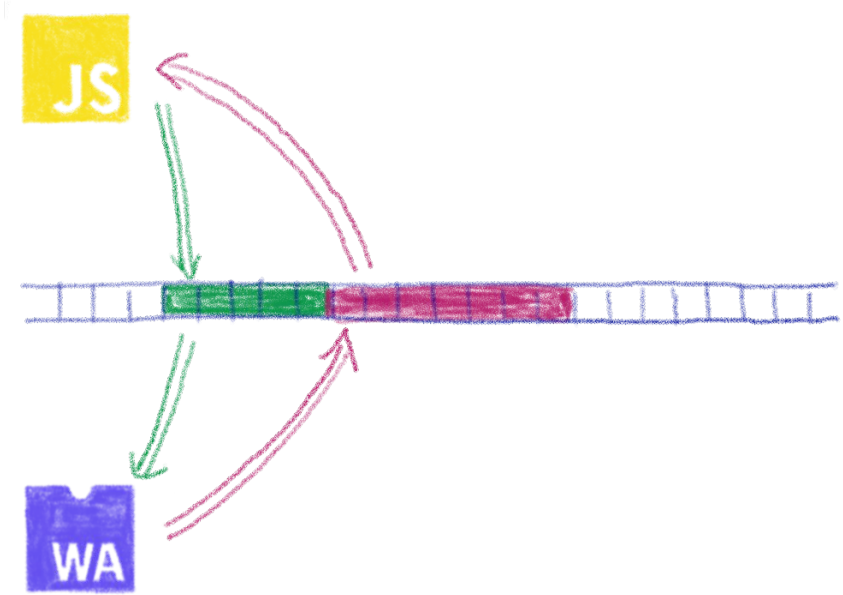
WASM currently has four available types:

- `i32`: 32-bit integer
- `i64`: 64-bit integer
- `f32`: 32-bit float
- `f64`: 64-bit float

Types from languages compiled to WASM are mapped to these types

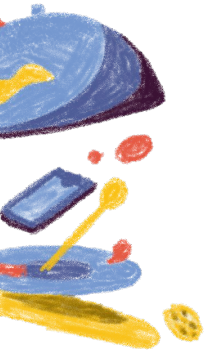
How can we share data?



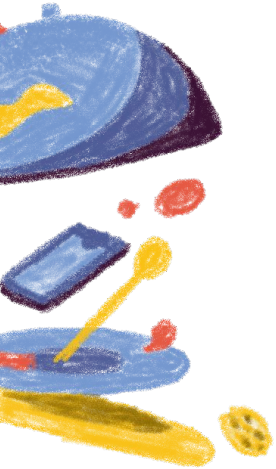


Using the same data in WASM and JS?
Shared linear memory between them!

You can do steps 03 and 04 now



Let's code, mates!

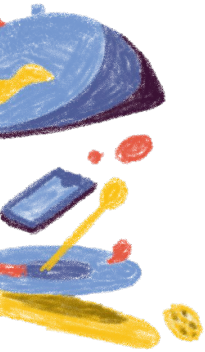


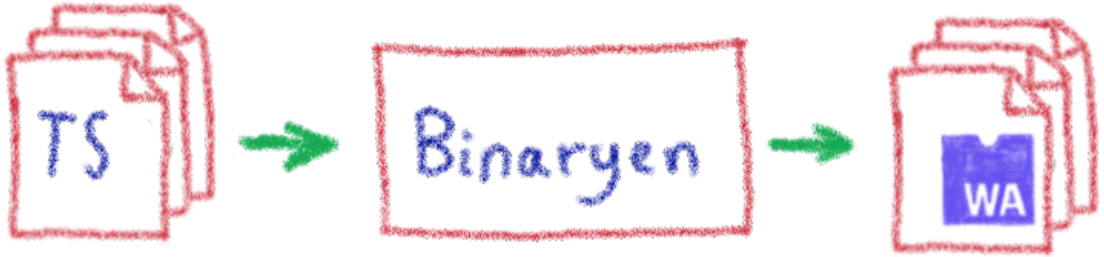
DEVFEST PARIS 2020

AssemblyScript

Writing WASM without learning a new language

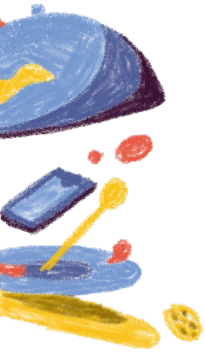
TypeScript subset compiled to WASM

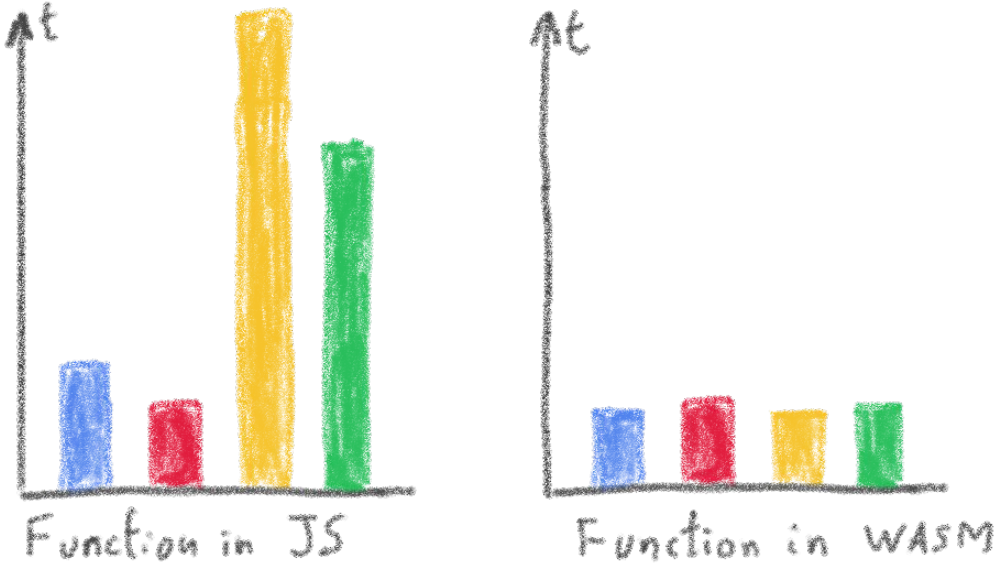




Why would I want to compile TypeScript to WASM?

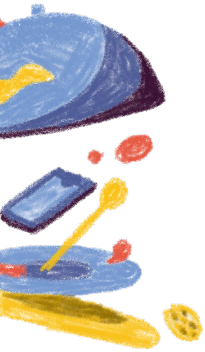
Ahead of Time compiled TypeScript

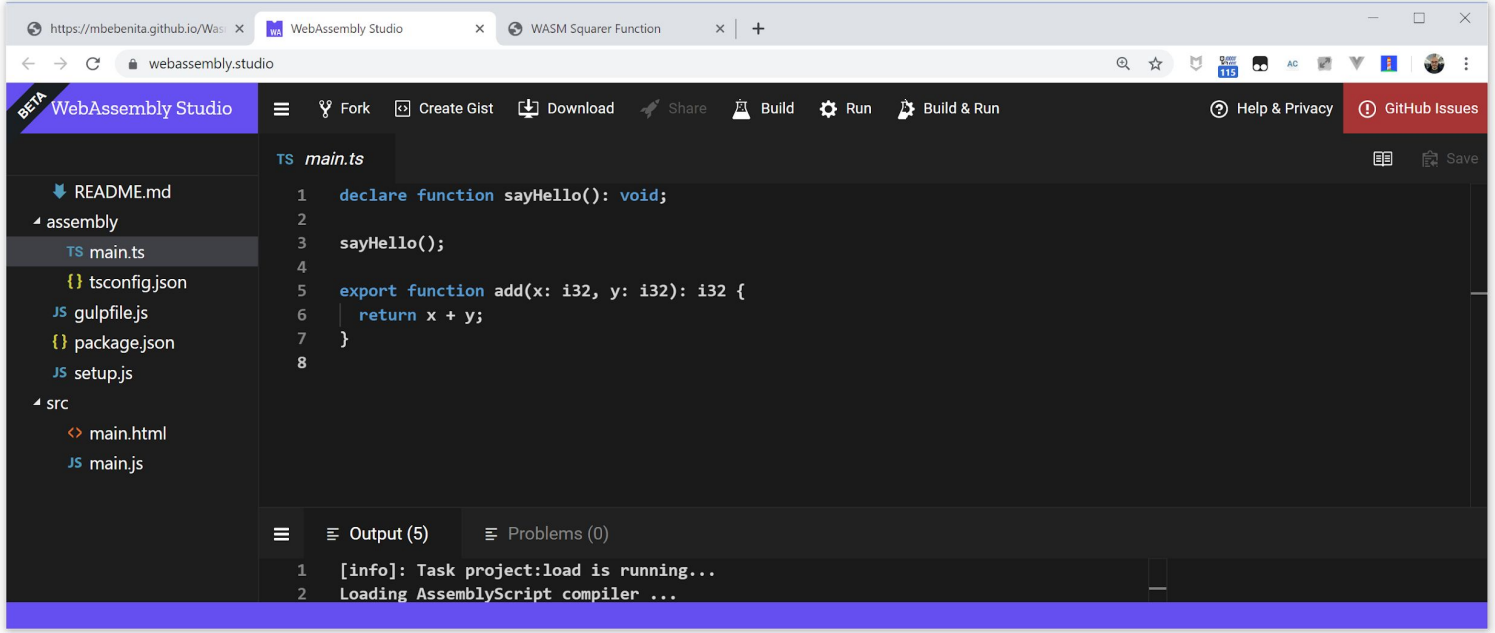


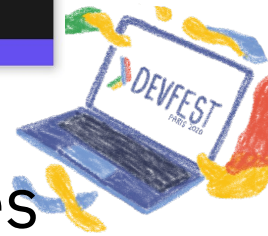


More predictable performance

Avoiding the dynamicness of JavaScript

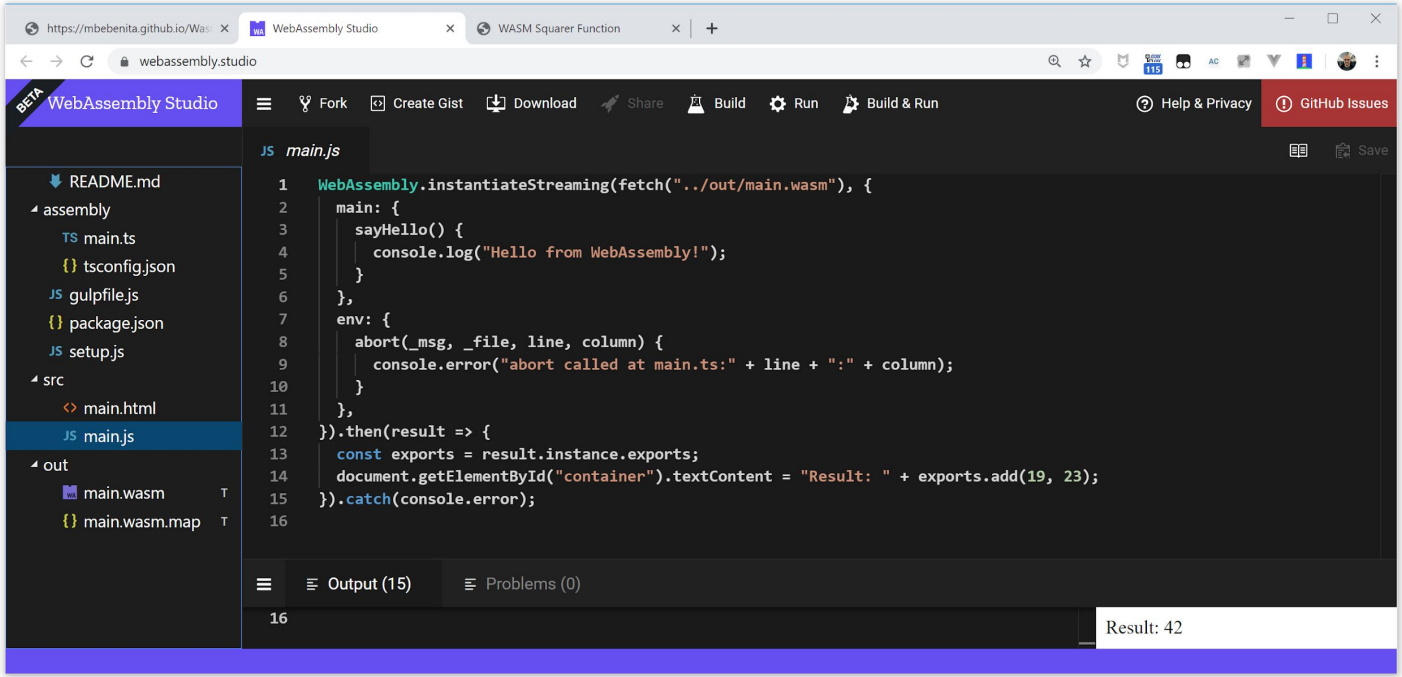






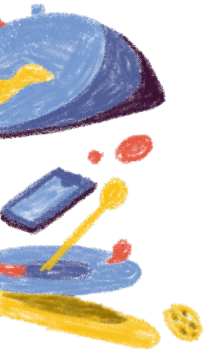
More specific integer and floating point types

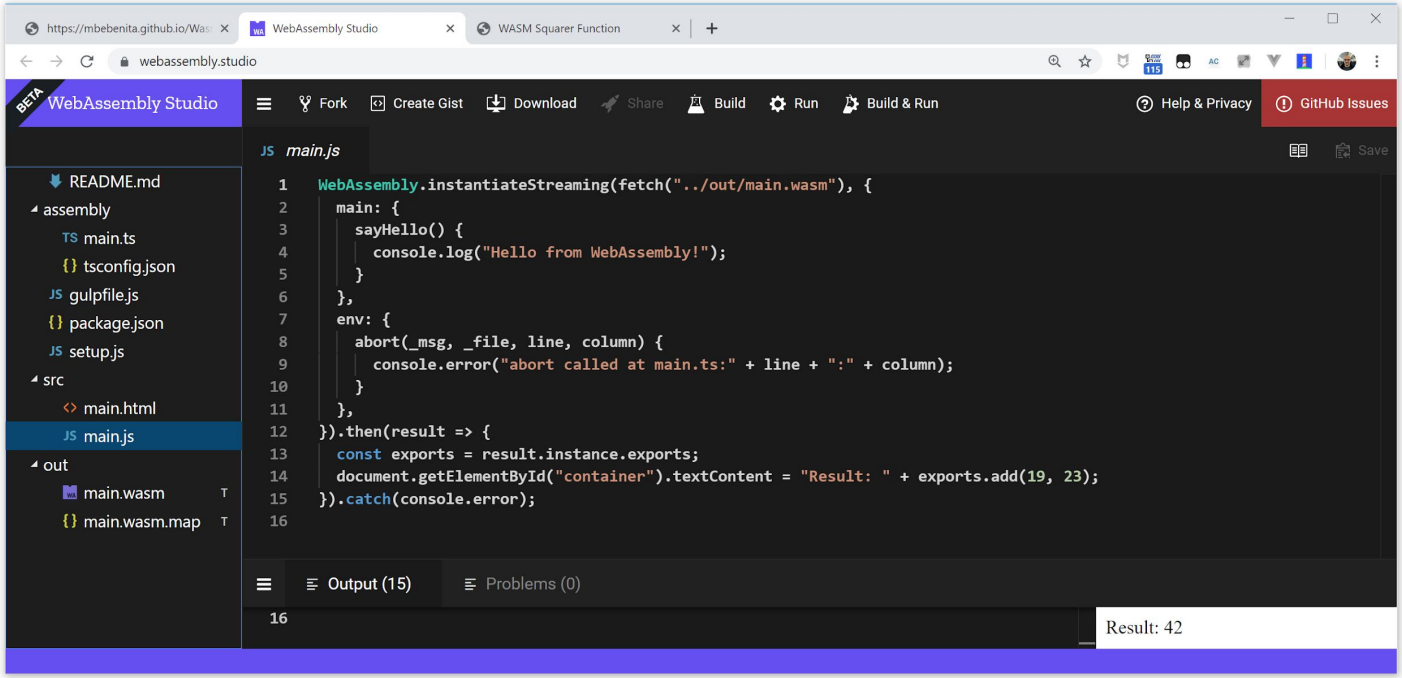
Objects cannot flow in and out of WASM yet



Using a loader to write/read them to/from memory

No direct access to DOM

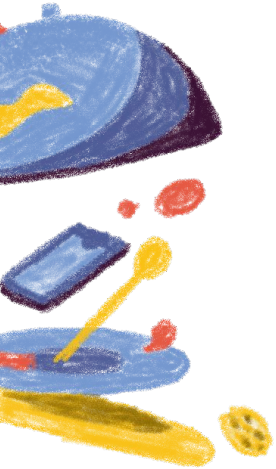


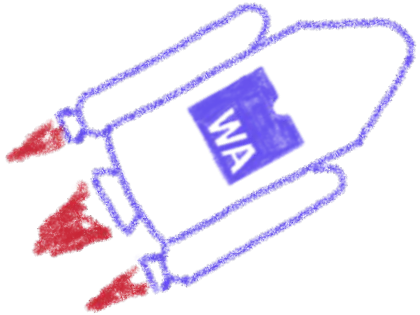


Glue code using exports/imports to/from JavaScript

You can do step 05 now

Let's code, mates!



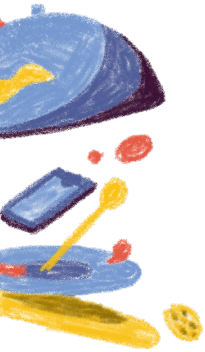


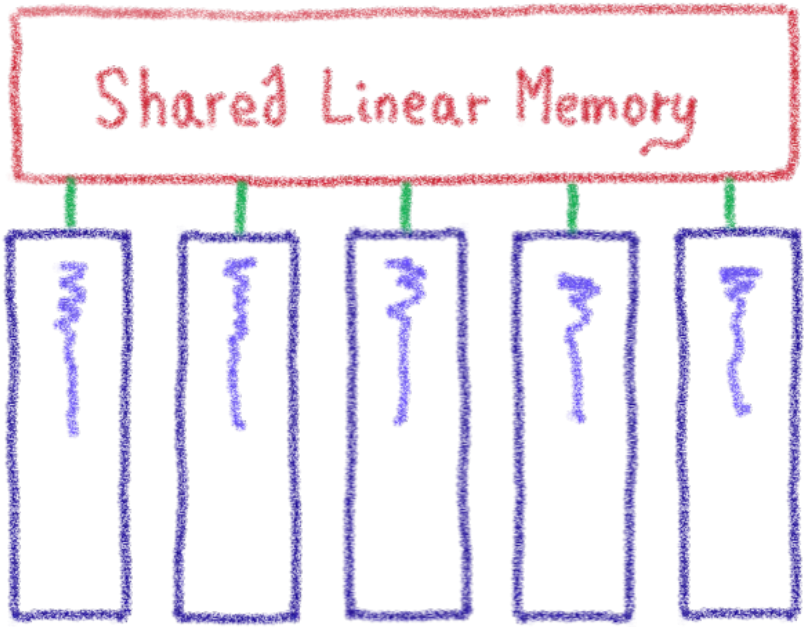
DEVFEST PARIS 2020

Future

To the infinity and beyond!

WebAssembly Threads

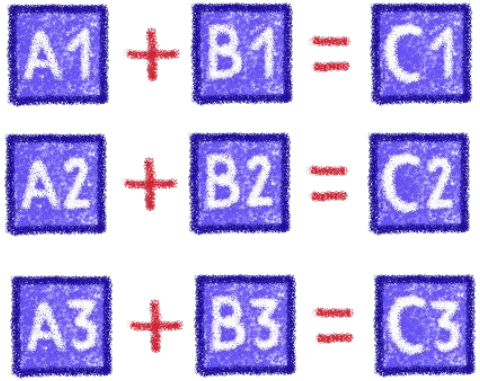




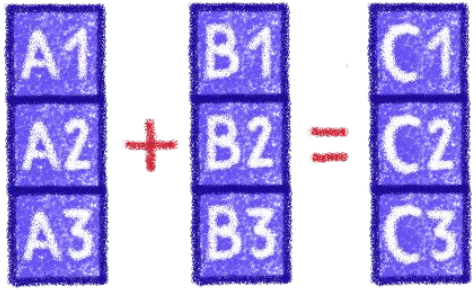
Threads on Web Workers with shared linear memory

SIMD

Multiple scalar operations



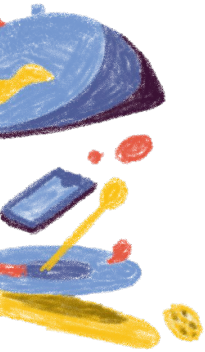
Single vectorial operation



Already available in Wasmer

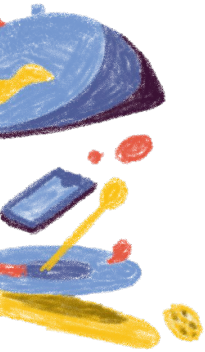
Single Instruction, Multiple Data

Garbage collector



And exception handling

WASI



WASI

WASI

The WebAssembly System Interface

WASI is a modular system interface for WebAssembly. As described in the initial announcement, it's focused on security and portability.

WASI is being standardized in a subgroup of the WebAssembly CG. Discussions happen in GitHub issues, pull requests, and bi-weekly Zoom meetings.

For a quick intro to WASI, including getting started using it, see the intro document.

The Wasmtime runtime's tutorial contains examples for how to target WASI from C and Rust. The resulting .wasm modules can be run in any WASI-compliant runtime.

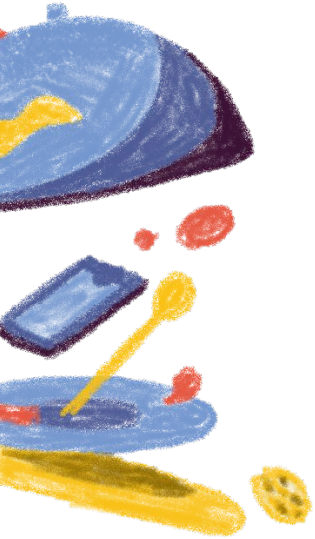
For more documentation, see the documents guide.

WebAssembly System Interface

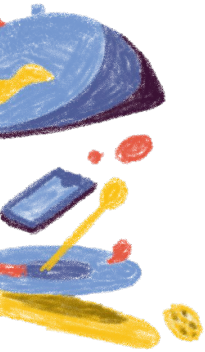
DEVFEST PARIS 2020

WebAssembly ❤ Web Components

How to hide the complexity and remove friction

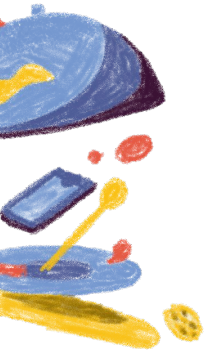


The 3 minutes context



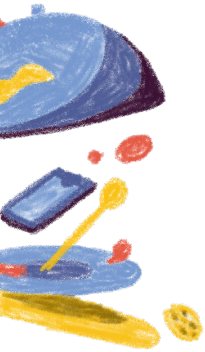
What the heck are web component?

Web Components



Web standard W3C

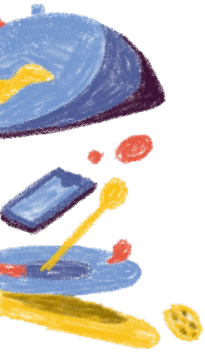
Web Components



Available in all modern browsers:

Firefox, Safari, Chrome

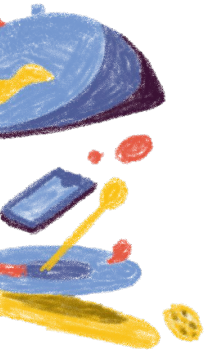
Web Components



Create your own HTML tags

Encapsulating look and behavior

Web Components



Fully interoperable

With other web components, with any framework

Web Components

CUSTOM ELEMENTS

SHADOW DOM

TEMPLATES

Custom Element

DEVFEST PARIS 2020

</> To define your own HTML tag

```
<body>
  ...
  <script>
    window.customElements.define('my-element',
        class extends HTMLElement {...});
  </script>
  <my-element></my-element>
</body>
```

Shadow DOM

To encapsulate subtree and style in an element


```
<button>Hello, world!</button>
<script>
var host = document.querySelector('button');
const shadowRoot = host.attachShadow({mode:'open'});
shadowRoot.textContent = 'こんにちは、影の世界!';
</script>
```

Template

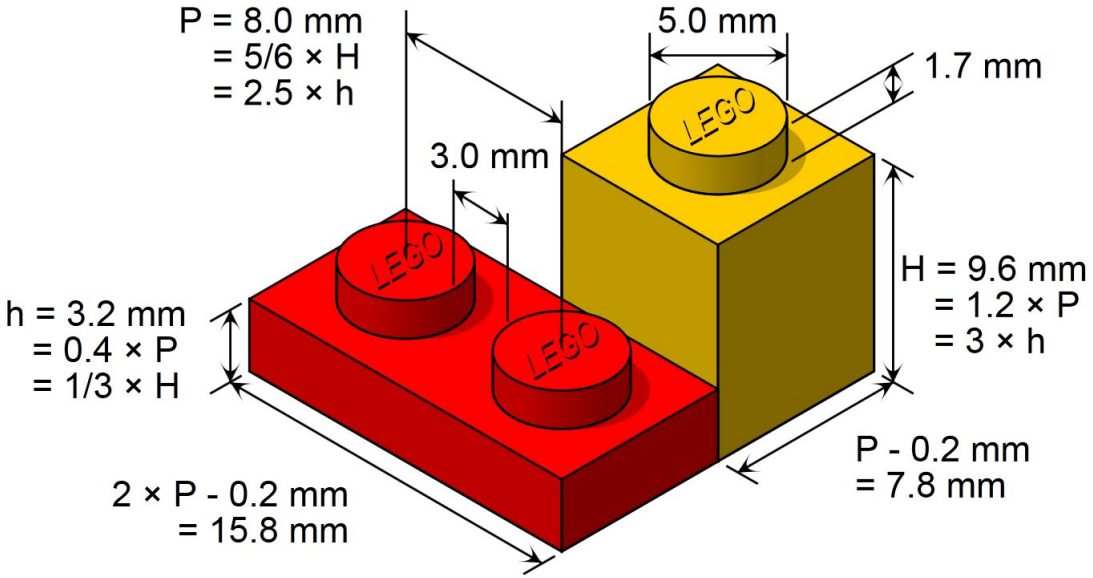
To have clonable document template

```
<template id="mytemplate">
  <img src="" alt="great image">
  <div class="comment"></div>
</template>

var t = document.querySelector('#mytemplate');
// Populate the src at runtime.
t.content.querySelector('img').src = 'logo.png';
var clone = document.importNode(t.content, true);
document.body.appendChild(clone);
```

But in fact, it's just an element...

- Attributes
- Properties
- Methods
- Events



You can do step 06 and 07 now

Let's code, mates!