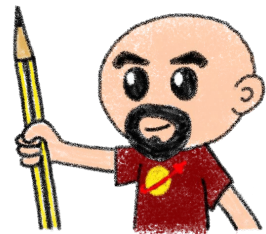




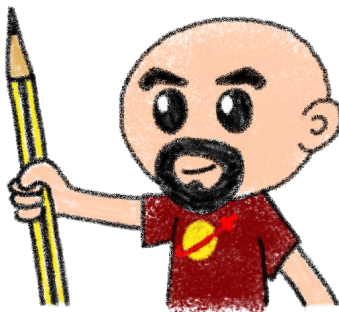
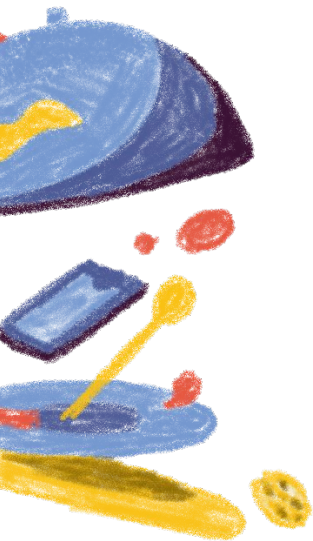
Hands on WebAssembly

Horacio Gonzalez
@LostInBrittany



Who are we?

Introducing myself and introducing ~~OVH~~ OVHcloud



Horacio Gonzalez



@LostInBrittany

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

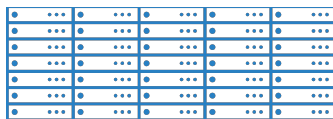


OVHcloud: A Global Leader

200k Private cloud
VMs running



**Dedicated IaaS
Europe**



Hosting capacity :
**1.3M Physical
Servers**

360k
Servers already
deployed




Own
20Tbps
Network
with
35 PoPs

30 Datacenters

> **1.3M** Customers in **138** Countries

OVHcloud: Our solutions




Cloud

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




Mobile Hosting

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



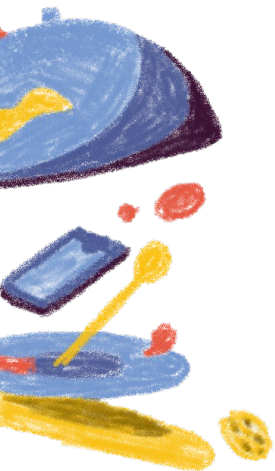
Web Hosting

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



Telecom

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



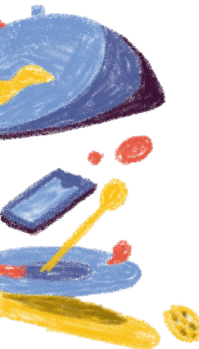
010
0101010
101101010
101010101

How is the codelab structured?

What are we coding today?



A GitHub repository



The screenshot shows a GitHub repository page for 'wasm-codelab' by 'LostInBrittany'. The repository is private and has 2 stars and 0 forks. It contains 18 commits, 1 branch, 0 releases, and 2 contributors. The commit history is as follows:

Commit	Message	Time ago
Latest commit	Game of Life working, even in local	1 hour ago
app	Step-02 and init Step-01	10 hours ago
scripts	Project README done	14 hours ago
step-01	Fixing typos	8 hours ago
step-02	Fixing links	8 hours ago
step-03	Fixing bugs and typos, adding offline version of step-04	6 hours ago
step-04	Fixing bugs and typos, adding offline version of step-04	6 hours ago
step-05	Game of Life working, even in local	1 hour ago
README.md	Fixing links	8 hours ago

The README.md file content is as follows:

DevFest Nantes 2019 WebAssembly Codelab

We have built this [WebAssembly Codelab](#) as a quick entry point to [WebAssembly](#).

What are the objectives of this tutorial

Follow the tutorial to learn the concepts behind WebAssembly (WASM), write your first WASM libraries, compile existing libraries to WASM and generally understand how WASM open new possibilities in the web development ecosystem.

What do I need to use this tutorial?

The tools strictly needed for this tutorial are a modern web browser (ideally [Chrome](#) or [Chromium](#)), a text editor (we suggest the excellent [Visual Studio Code](#)), [Node JS](#), and a web-server to test your code.



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

@LostInBrittany



Nothing to install

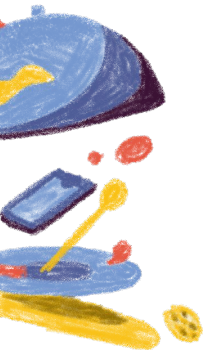


C++11 -Os	COMPILE	Wat	ASSEMBLE	DOWNLOAD	Firefox x86 Assembly
<pre>1 int squarer(int num) { 2 return num * num; 3 }</pre>		<pre>1 (module 2 (type \$type0 (func (param i32) 3 (result i32))) 4 (table 0 anyfunc) 5 (memory 1) 6 (export "memory" memory) 7 (export "_Z7squarer_i" \$func0) 8 (func \$func0 (param \$var0 i32) 9 (result i32) 10 get_local \$var0 11 get_local \$var0 12 i32.mul 13) 14)</pre>			<pre>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</pre>

Using WebAssembly Explorer
and WebAssembly Studio



Only additional tool: a web server



 python™

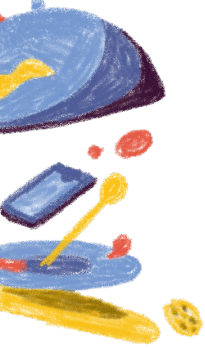




Because of the browser security model



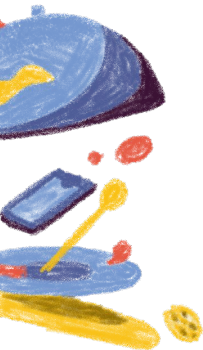
Procedure: follow the steps



Step by step



But before coding, let's speak



What's this WebAssembly thing?





Did we say WebAssembly?

WASM for the friends...



WebAssembly, what's that?



What's WASM?

Does it replace JS?

I code webapps in Rust?

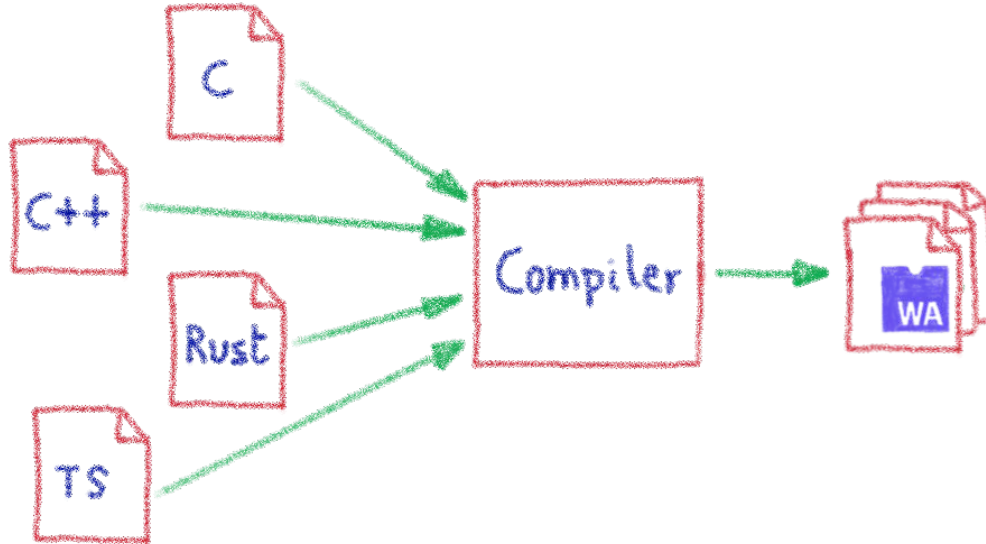
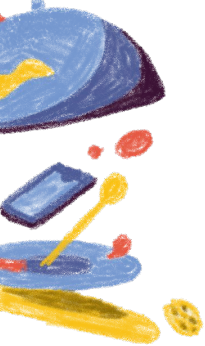


Is HTML/CSS/JS stack obsolete?

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



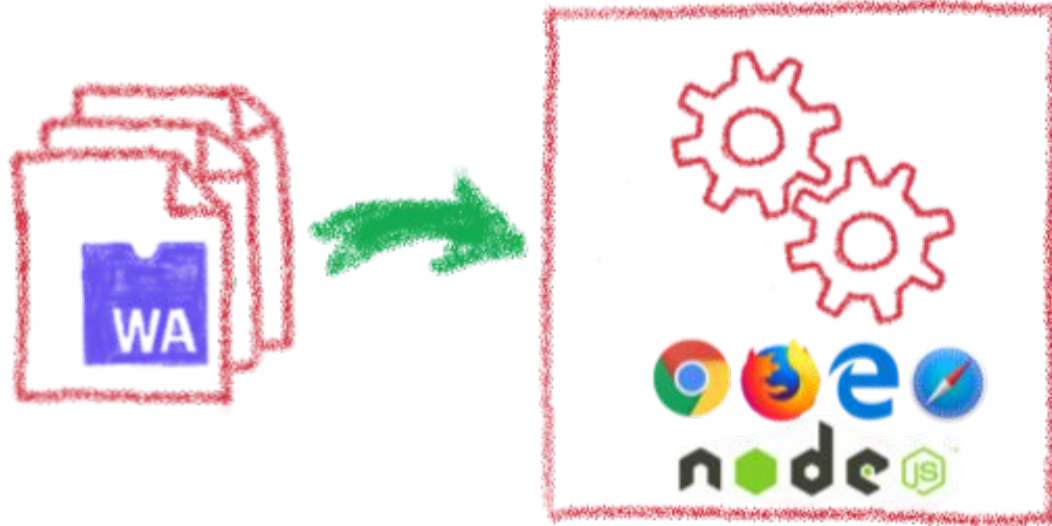
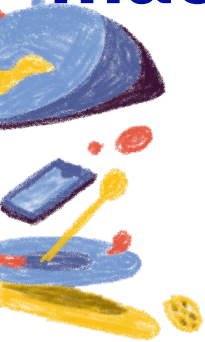
A low-level binary format for the web



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

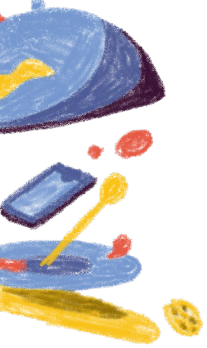
Fast & Efficient ⚡

🔒 Memory-safe & Sandboxed

Open & Debuggable 📄

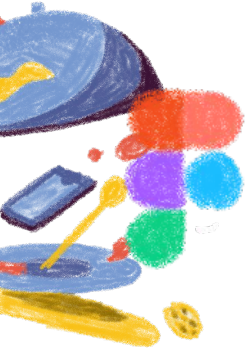
WWW Part of the Web Platform

But above all...



WebAssembly is not meant to replace JavaScript

Who is using WebAssembly today?

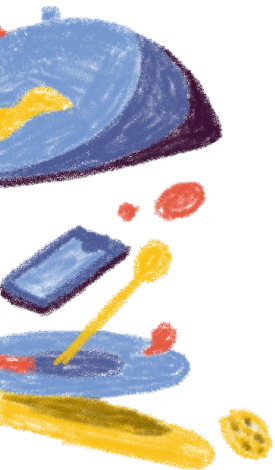


FIGMA

AUTOCAD



And many more others...



W3C

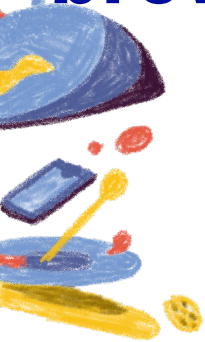


A bit of history

Remembering the past
to better understand the present



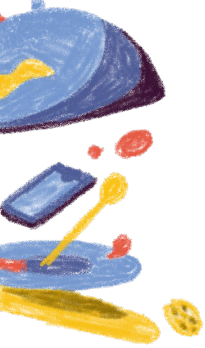
Executing other languages in the browser



A long story, with many failures...



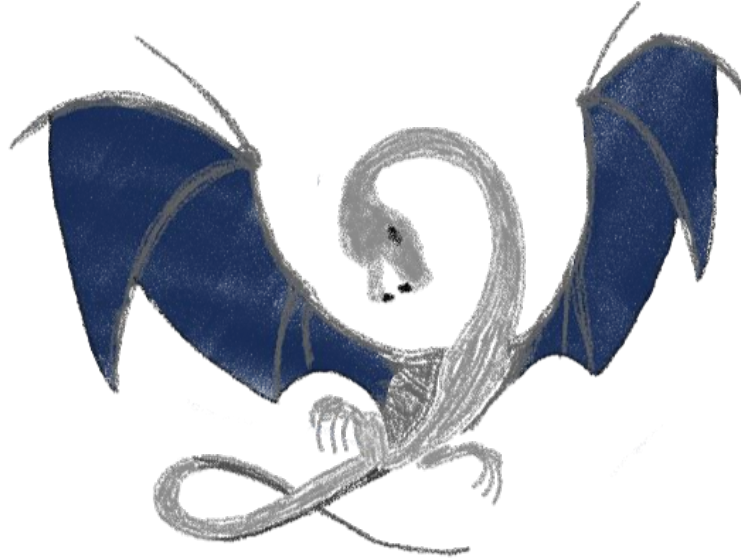
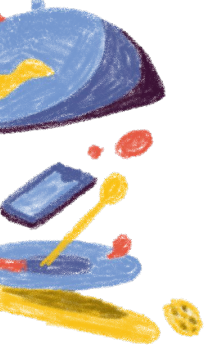
2012 - From C to JS: enter emscripten



Passing by LLVM pivot

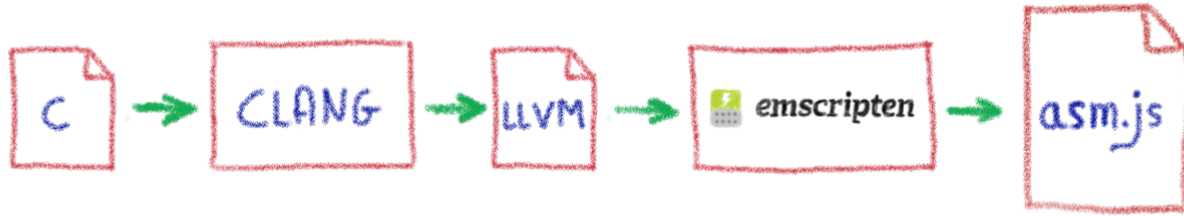
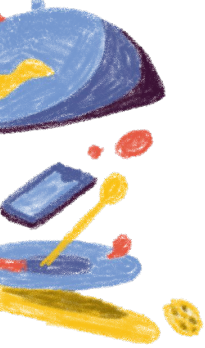


Wait, dude! What's LLVM?



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

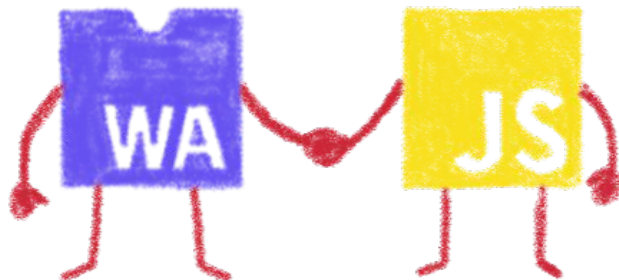
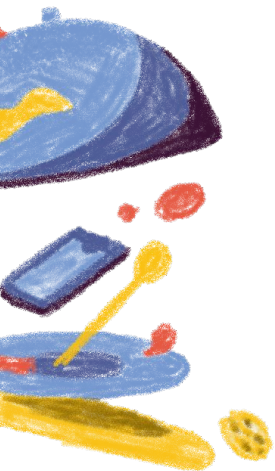
moz://a

Google

Joint effort



W3C



Hello W(ASM)orld

My first WebAssembly program



Do you remember your 101 C course?



```
1  #include <stdio.h>
2
3  int main(int argc, char ** argv) {
4  |  printf("Hello, world!\n");
5  |  }
6  |
```

A simple *HelloWorld* in C

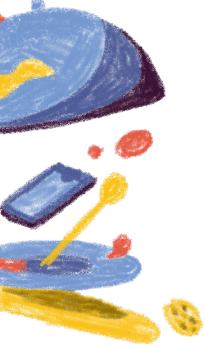


We compile it with emscripten

```
horacio@DESKTOP-6KHP1S2: ~/git/wasm/hello_world × horacio@DESKTOP-6KHP1S2: ~/git/emscripten × + ▾
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...

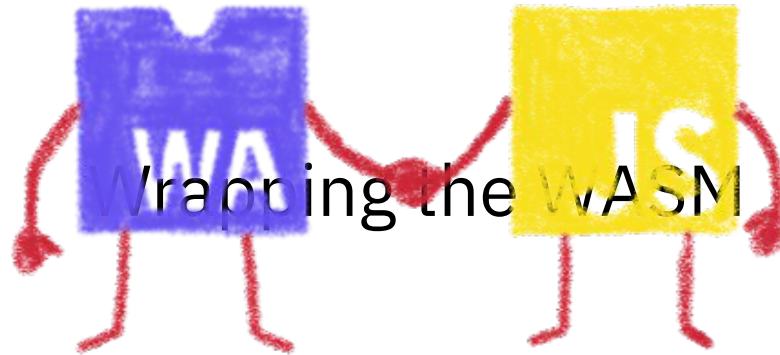


01011010
01101010
1011001101

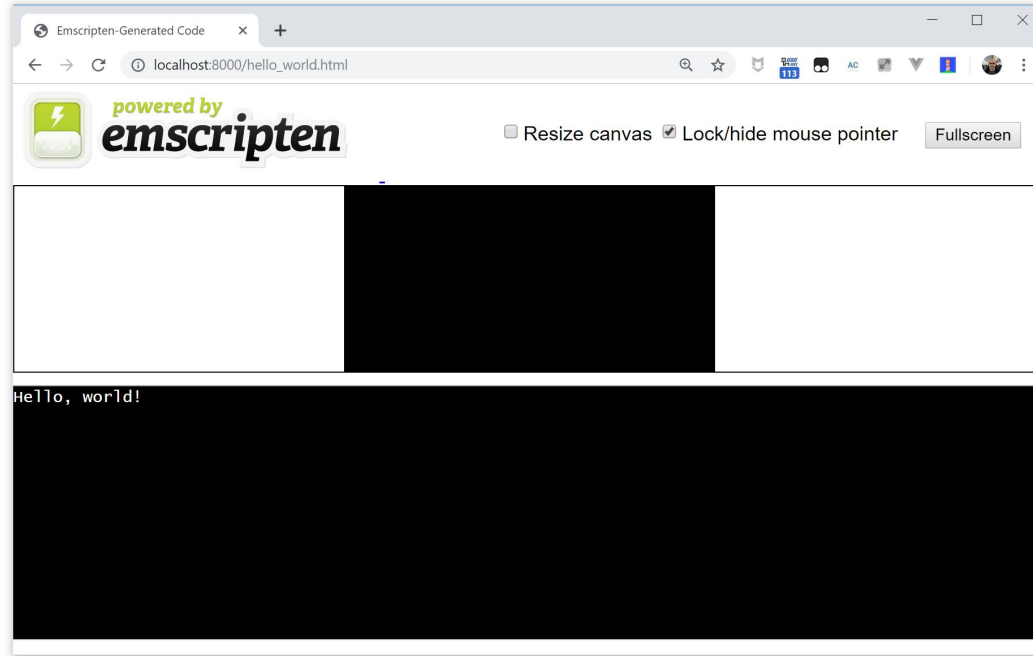
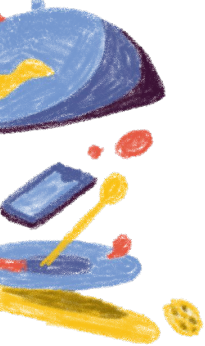


Binary file, in the binary WASM format

We also get a .js file...



And a .html file



To quickly execute in the browser our WASM

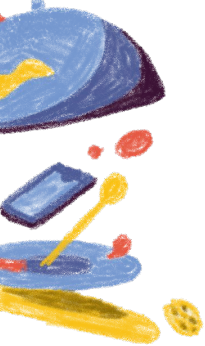
And in a more Real World™ case?

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...



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

WebAssembly Explorer v2.18

Please also check out WebAssembly Studio

Options: C++11 -Os, COMPILE, Wat, ASSEMBLE, DOWNLOAD, Firefox x86 Assembly

Auto Compile (checked)
LLVM x86 Assembly (unchecked)
Examples (dropdown)
C++11 (dropdown)
Optimization Level (dropdown)
Fast Math (unchecked)
No Inline (unchecked)
No RTTI (unchecked)
No Exceptions (unchecked)
Clean WAT (unchecked)
Baseline JIT (unchecked)

OPEN IN WASMFIDDLE

```
1 Welcome to the WebAssembly Explorer
2 =====
3
4 Here you can translate C/C++ to WebAssembly, and then see the machine code generated by the browser.
5
6 For bugs, comments and suggestions see: https://github.com/mbebenita/WasmExplorer
7 Built with Clang/LLVM, AngularJS, Ace Editor, Emscripten, SpiderMonkey, Binaryen and Capstone.js.
8
9
10 Service version 3.5 (js: JavaScript-C55.0a1; clang: 5.0.0 (https://chromium.googlesource.com/external/github.com/llvm-mirror...))
11
```

Let's use WASM Explorer

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



Let's begin with the a simple function



C++11 -Os	COMPILE	Wat	ASSEMBLE	DOWNLOAD	Firefox x86 Assembly
<pre>1 int squarer(int num) { 2 return num * num; 3 }</pre>		<pre>1 (module 2 (type \$type0 (func (param i32) 3 (result i32))) 4 (table 0 anyfunc) 5 (memory 1) 6 (export "memory" memory) 7 (export "_Z7squareri" \$func0) 8 (func \$func0 (param \$var0 i32) 9 (result i32) 10 get_local \$var0 11 get_local \$var0 12 i32.mul 13) 14)</pre>			<pre>- 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</pre>

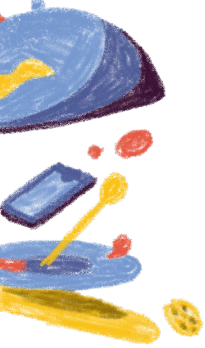
WAT: WebAssembly Text Format

Human readable version of the .wasm binary

@LostInBrittany



Download the binary .wasm file



01011010
01101101
10110011
101101



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

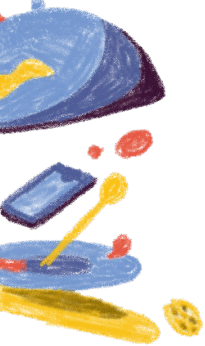


Instantiating the WASM

```
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();
20
```

Loading the squarer function

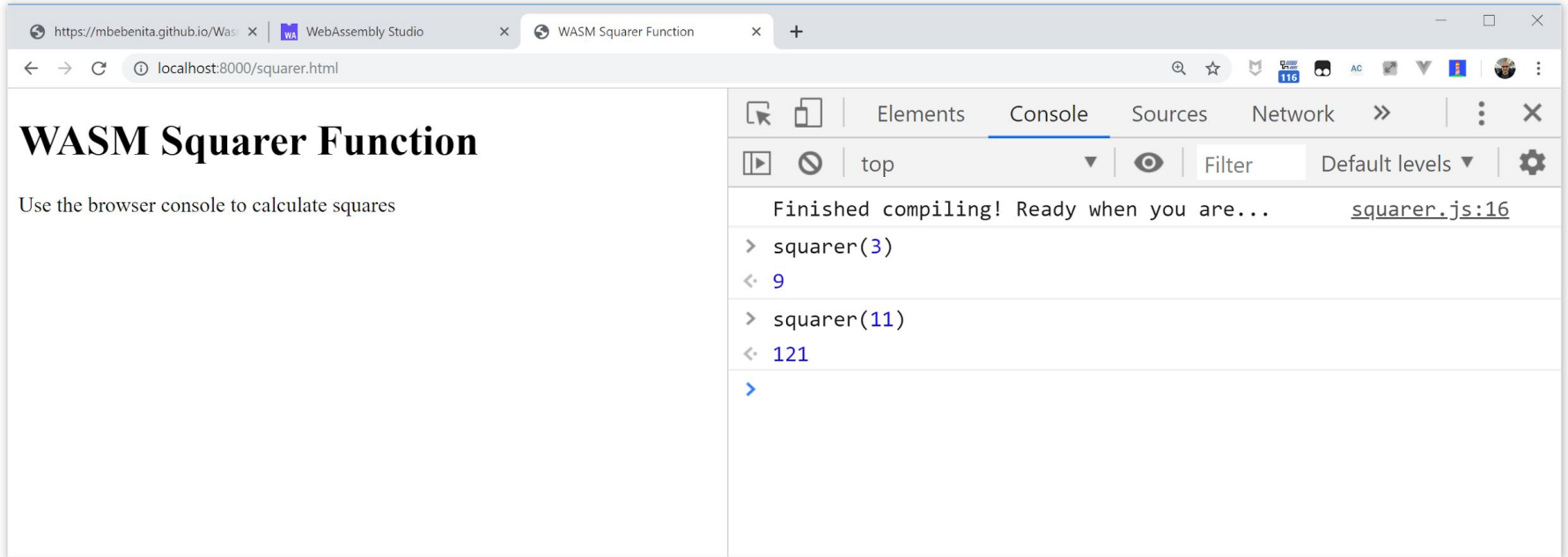


```
wasm > squarer > <> squarer.html > ...
 1 <!DOCTYPE html>
 2 <html>
 3 <head>
 4   <meta charset="utf-8" />
 5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
 6   <title>WASM Squarer Function</title>
 7   <meta name="viewport" content="width=device-width, initial-scale=1">
 8 </head>
 9 <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 x WASM Squarer Function x +

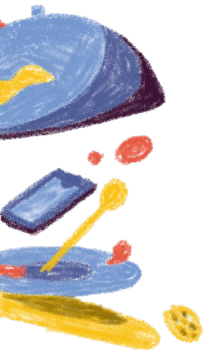
localhost:8000/squarer.html

WASM Squarer Function

Use the browser console to calculate squares

```
Finished compiling! Ready when you are... squarer.js:16
> squarer(3)
< 9
> squarer(11)
< 121
>
```


You can do steps 01 and 02 now



Let's code, mates!

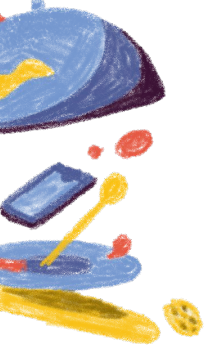


WASM outside the browser

Not only for web developers



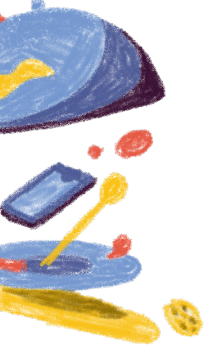
Run any code on any client... almost



Languages compiling to WASM



Includes WAPM



wapm install optipng

Oh, like npm for WASM!

The WebAssembly Package Manager



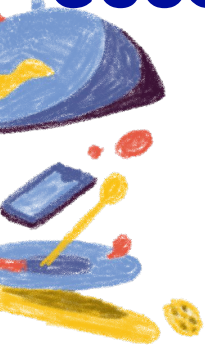


Some use cases

What can I do with it?



Tapping into other languages ecosystems



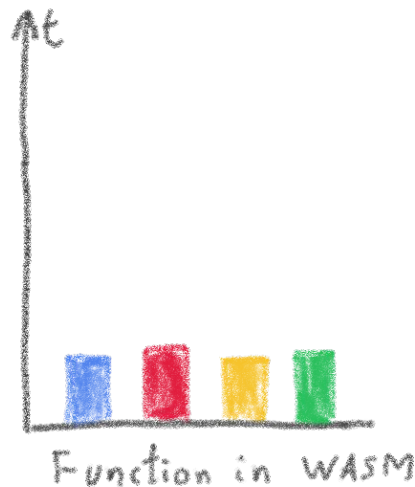
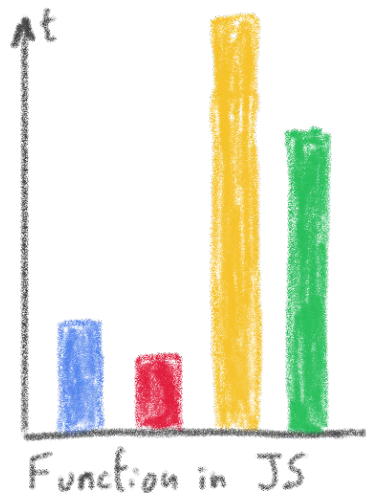
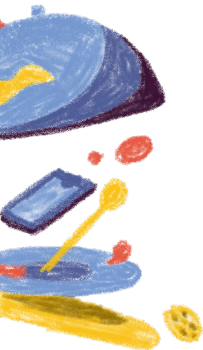
SQUOSH.APP

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



Don't rewrite libs anymore

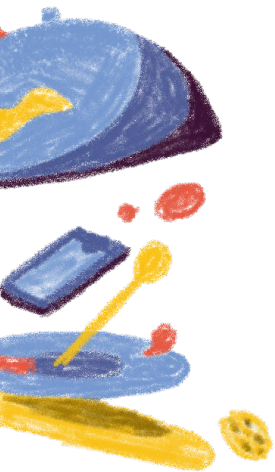
Replacing problematic JS bits



- Browser 1
- Browser 2
- Browser 3
- Browser 4

Predictable performance
Same peak performance, but less variation





Communicating between JS and WASM

Shared memory, functions...



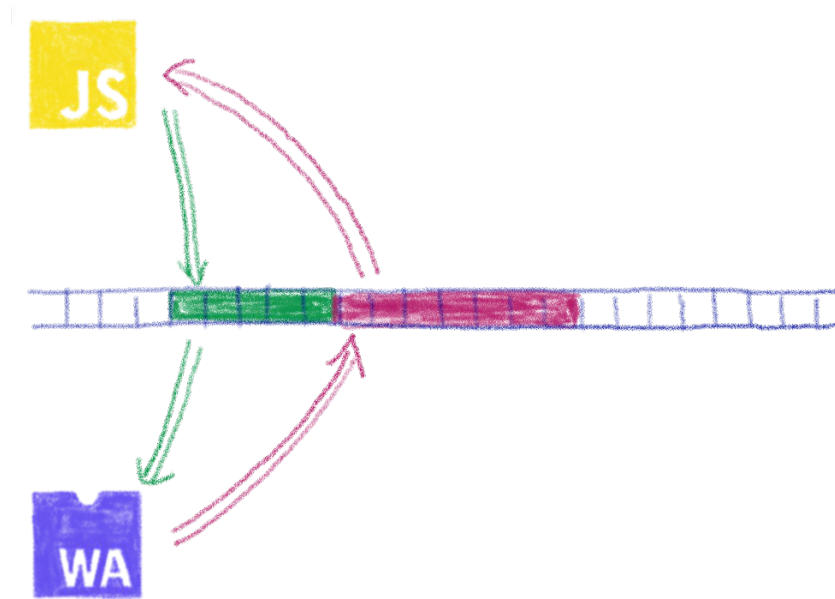
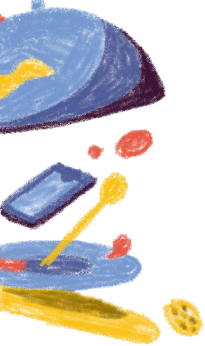
Native WASM types are limited

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

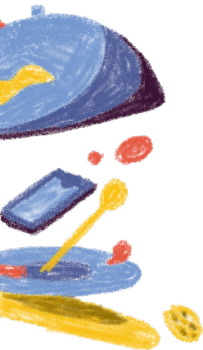
How can we share data?



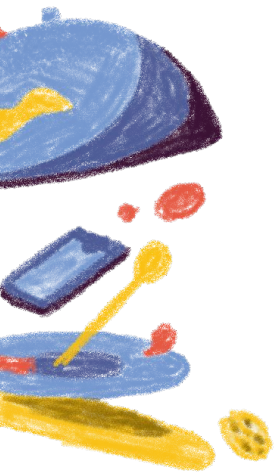
Using the same data in WASM and JS?
Shared linear memory between them!



You can do steps 03 and 04 now



Let's code, mates!

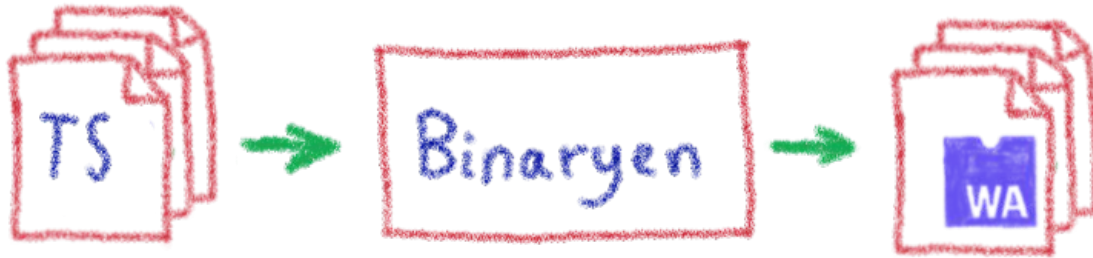
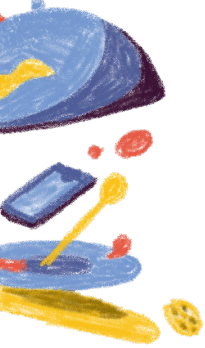


AssemblyScript

Writing WASM without learning a new language



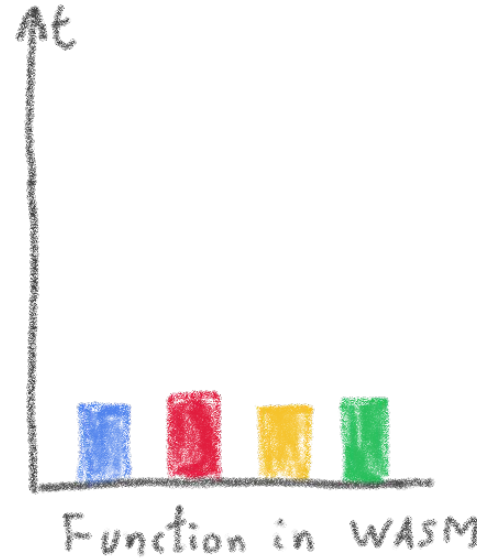
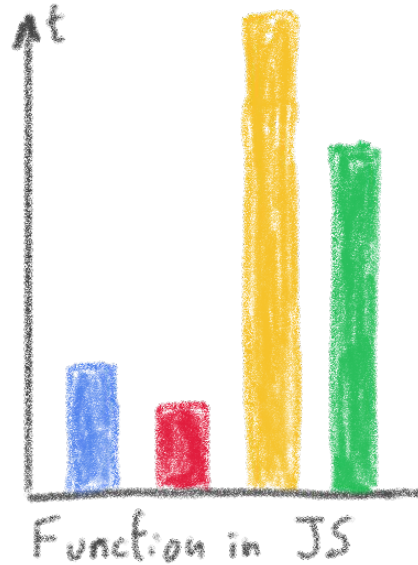
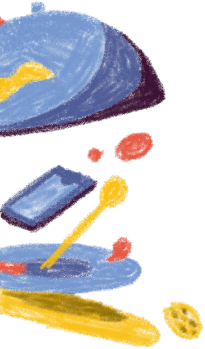
TypeScript subset compiled to WASM



Why would I want to compile TypeScript to WASM?



Ahead of Time compiled TypeScript

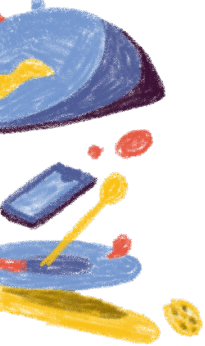


- Browser 1
- Browser 2
- Browser 3
- Browser 4

More predictable performance



Avoiding the dynamicness of JavaScript



```
1 declare function sayHello(): void;
2
3 sayHello();
4
5 export function add(x: i32, y: i32): i32 {
6   return x + y;
7 }
8
```

Output (5) Problems (0)

```
1 [info]: Task project:load is running...
2 Loading AssemblyScript compiler ...
```

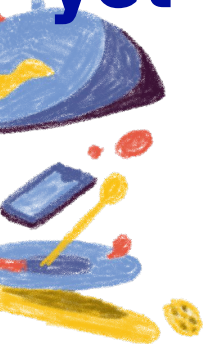


More specific integer and floating point types

Objects cannot flow in and out of WASM



yet



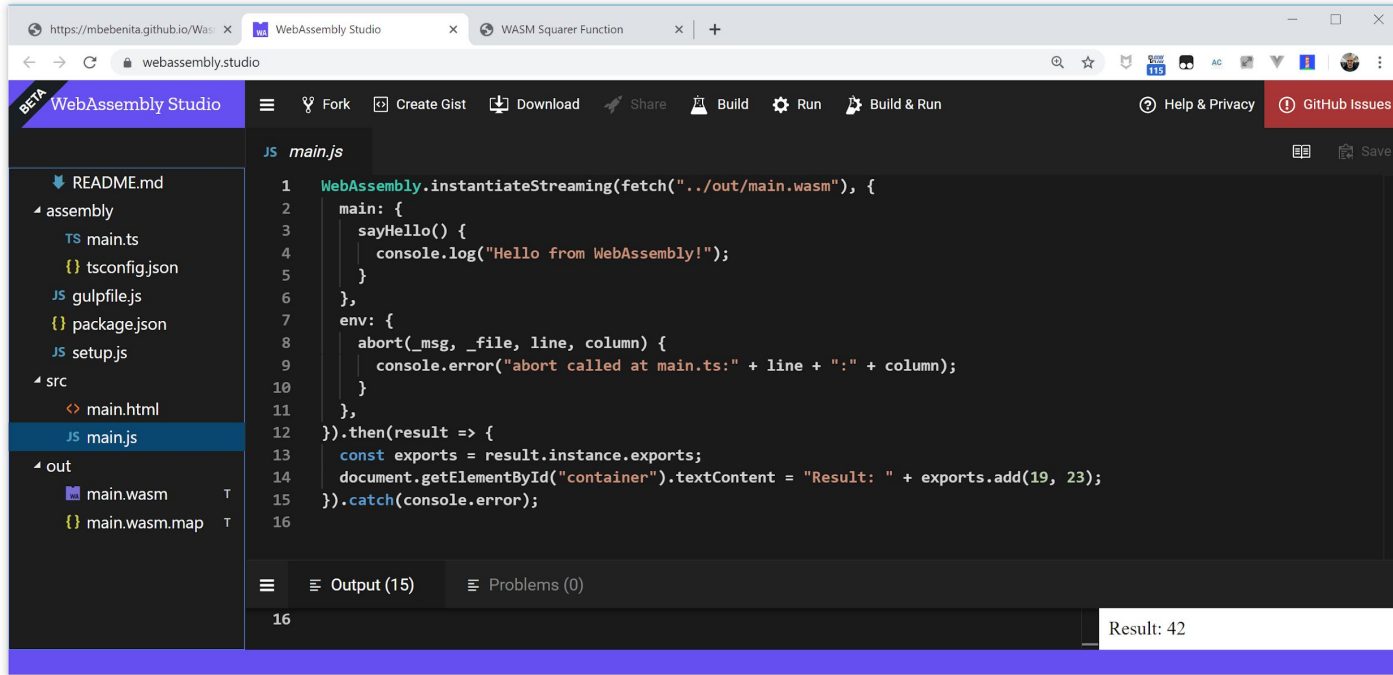
```
1 WebAssembly.instantiateStreaming(fetch("../out/main.wasm"), {
2   main: {
3     sayHello() {
4       console.log("Hello from WebAssembly!");
5     }
6   },
7   env: {
8     abort(_msg, _file, line, column) {
9       console.error("abort called at main.ts:" + line + ":" + column);
10    }
11  },
12 }).then(result => {
13   const exports = result.instance.exports;
14   document.getElementById("container").textContent = "Result: " + exports.add(19, 23);
15 }).catch(console.error);
16
```

Result: 42



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

No direct access to DOM

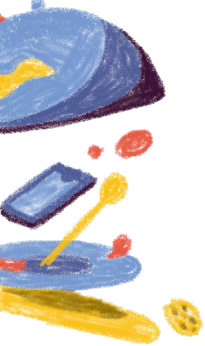


```
1 WebAssembly.instantiateStreaming(fetch("../out/main.wasm"), {
2   main: {
3     sayHello() {
4       console.log("Hello from WebAssembly!");
5     }
6   },
7   env: {
8     abort(_msg, _file, line, column) {
9       console.error("abort called at main.ts:" + line + ":" + column);
10    }
11  },
12 }).then(result => {
13   const exports = result.instance.exports;
14   document.getElementById("container").textContent = "Result: " + exports.add(19, 23);
15 }).catch(console.error);
16
```

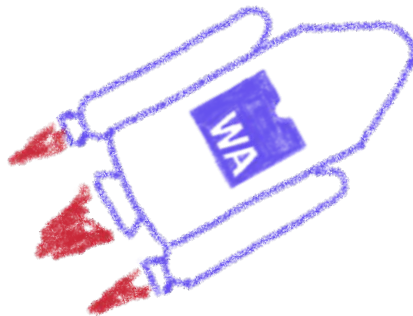
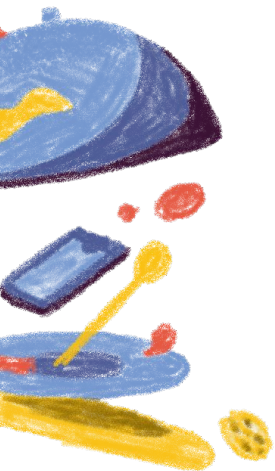
Result: 42

Glue code using exports/imports to/from JavaScript

You can do step 05 now



Let's code, mates!

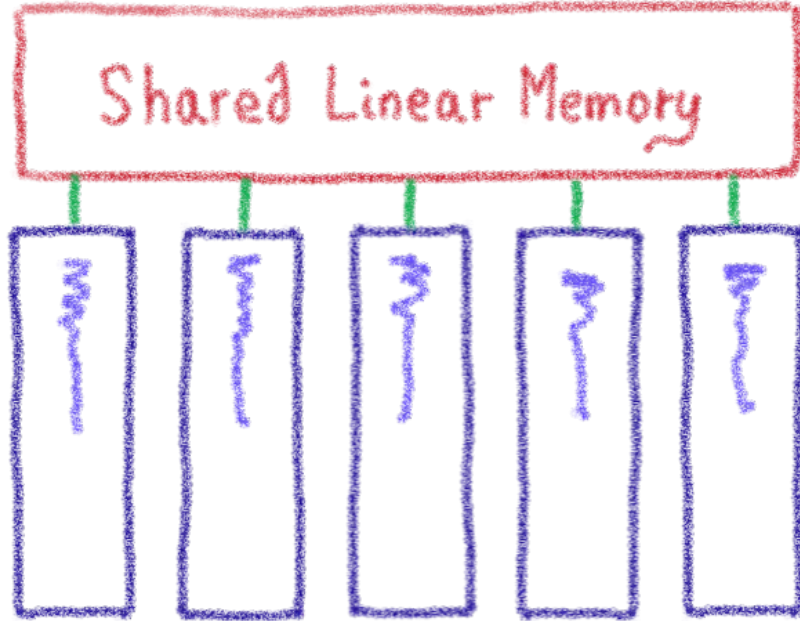
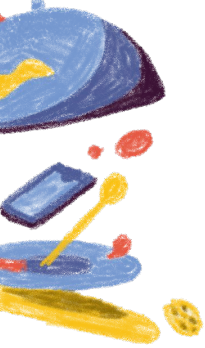


Future

To the infinity and beyond!



WebAssembly Threads



Threads on Web Workers with shared linear memory

SIMD

Multiple scalar operations

$$\begin{array}{l} \boxed{A1} + \boxed{B1} = \boxed{C1} \\ \boxed{A2} + \boxed{B2} = \boxed{C2} \\ \boxed{A3} + \boxed{B3} = \boxed{C3} \end{array}$$

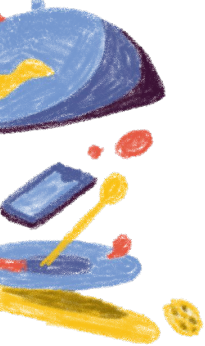
Single vectorial operation

$$\begin{array}{l} \boxed{A1} \\ \boxed{A2} \\ \boxed{A3} \end{array} + \begin{array}{l} \boxed{B1} \\ \boxed{B2} \\ \boxed{B3} \end{array} = \begin{array}{l} \boxed{C1} \\ \boxed{C2} \\ \boxed{C3} \end{array}$$

Already available
in  Wasmer

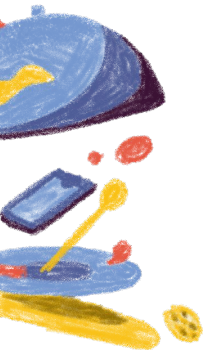
Single Instruction, Multiple Data

Garbage collector



And exception handling





The screenshot shows a blue webpage for WASI. At the top left is a small 'WASI' logo. At the top right is the text 'WASI' followed by a refresh icon. The main heading is 'WASI' in large white letters, with the subtitle 'The WebAssembly System Interface' below it. The page contains four paragraphs of text, each starting with a link to a specific document or resource. The text is white on a blue background.

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.

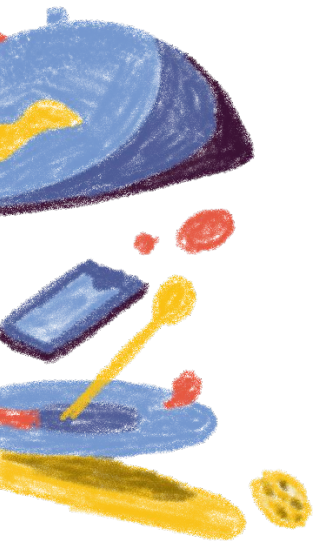
For more documentation, see [the documents guide](#).



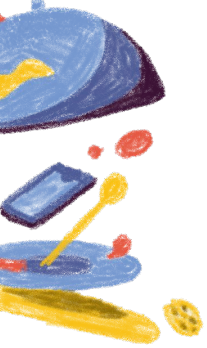
WebAssembly System Interface

WebAssembly Web Components

How to hide the complexity and remove friction



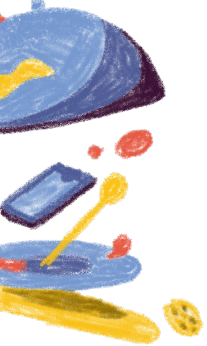
The 3 minutes context



What the heck are web component?



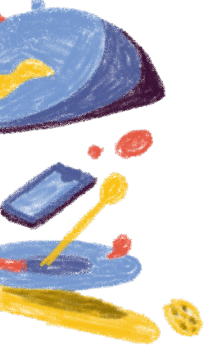
Web Components



Web standard W3C



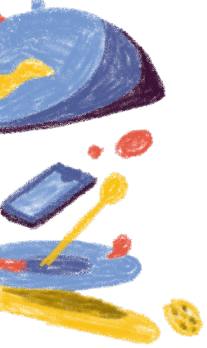
Web Components



Available in all modern browsers:
Firefox, Safari, Chrome



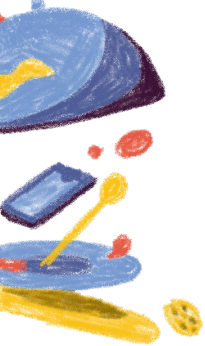
Web Components



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



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>
```

Hello, world!



こんにちは、影の世界!



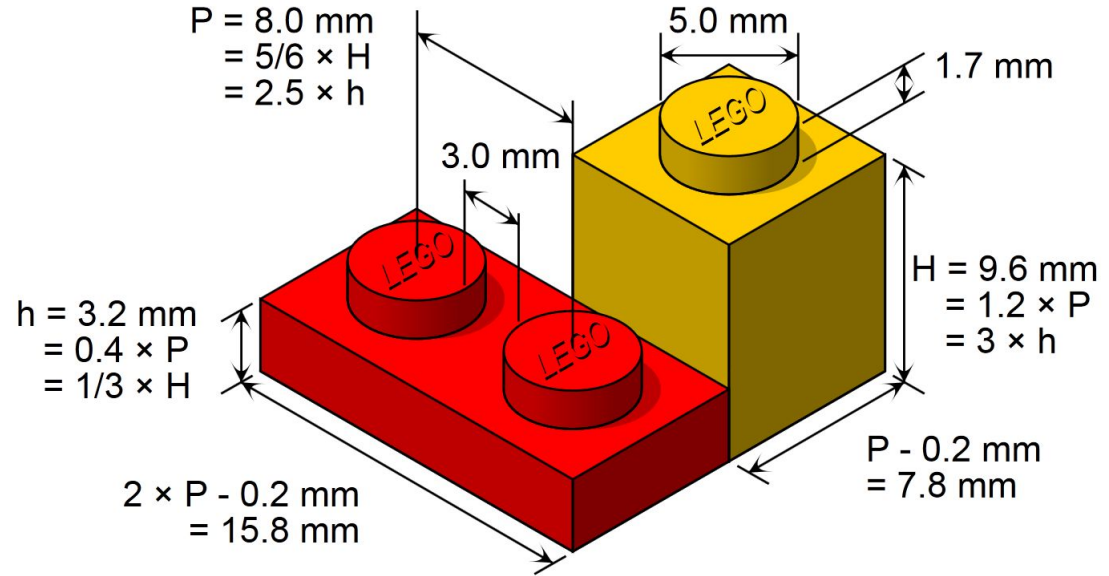
To have clonable document template

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

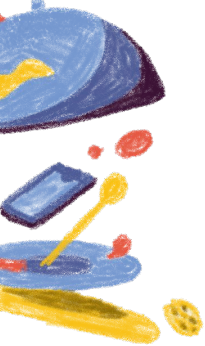
```
var t = document.querySelector('#mytemplate');  
// Populate the src at runtime.  
t.content.querySelector('img').src = 'logo.png';  
var clone = document.importNode(t.content, true);  
document.body.appendChild(clone);
```

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

- Attributes
- Properties
- Methods
- Events



You can do step 06 and 07 now



Let's code, mates!