Jug Summer Camp

-enjoy it-

Hands on Web Assembly

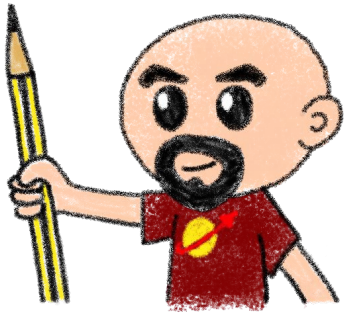
Speaker : Horacio Gonzalez - @LostInBrittany

SERLI

POITOU CHARENTES

Who are we?

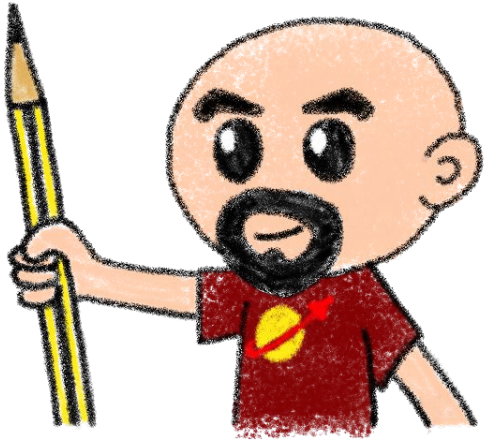
Introducing myself and introducing ~~OVH~~ OVHcloud



Horacio Gonzalez

@LostInBrittany

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



DevFest du Bout du Monde

Finist Devs

OVHcloud

DevRel Leader

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

Cloud

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

Mobile Hosting

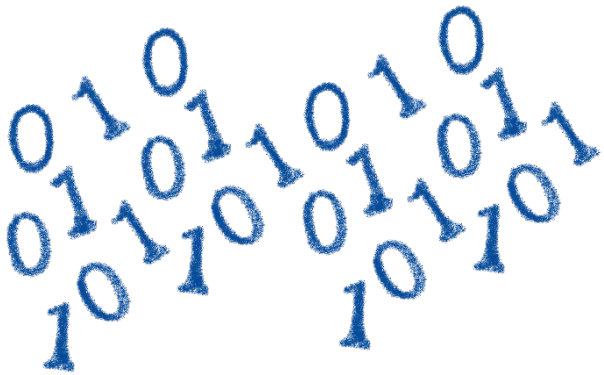
- Containers
- Compute
- Database
- Object Storage
- Securities
- Messaging

Web Hosting

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

Telecom

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



How is the codelab structured?

What are we coding today?

A GitHub repository

Search or jump to... / Pull requests Issues Marketplace Explore

LostInBrittany / wasm-codelab Private

Unwatch 2 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Security Insights Settings

A repository for the WASM codelab at DevFest Nantes 2019 Edit

Manage topics

18 commits 1 branch 0 releases 2 contributors

Branch: master New pull request Create new file Upload files Find file Clone or download

LostInBrittany Game of Life working, even in local Latest commit 28b108b 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

README.md

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

Nothing to install

C++11 -Os | COMPILE

```
1 int squarer(int num) {
2   return num * num;
3 }
```

Wat | ASSEMBLE | DOWNLOAD

```
1  (module
2    (type $type0 (func (param i32)
       (result i32)))
3    (table 0 anyfunc)
4    (memory 1)
5    (export "memory" memory)
6    (export "_Z7squareri" $func0)
7    (func $func0 (param $var0 i32)
       (result i32)
8      get_local $var0
9      get_local $var0
10     i32.mul
11   )
12 )
```

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

What's this WebAssembly thing?

Did we say WebAssembly?

WASM for the friends...

WebAssembly, what's that?

Can I code webapps in Rust?

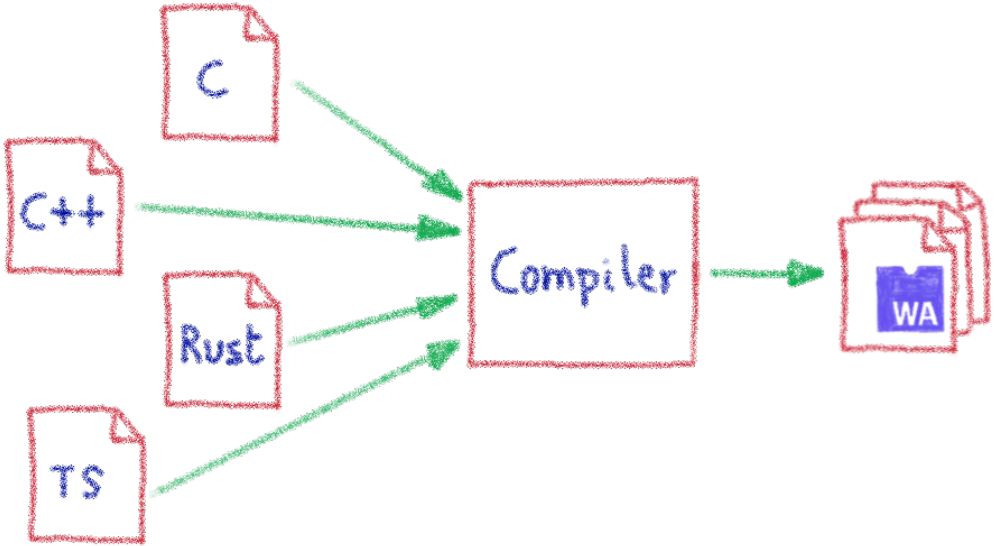
What's WASM ?

Does it replace JS ?

Is HTML/CSS/JS stack obsolete?

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

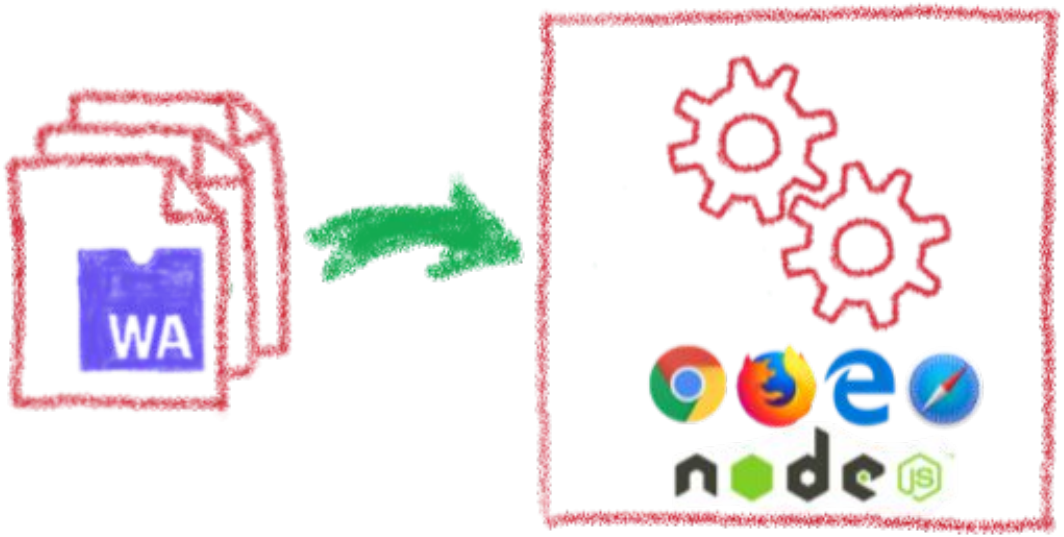
A low-level binary format for the web



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

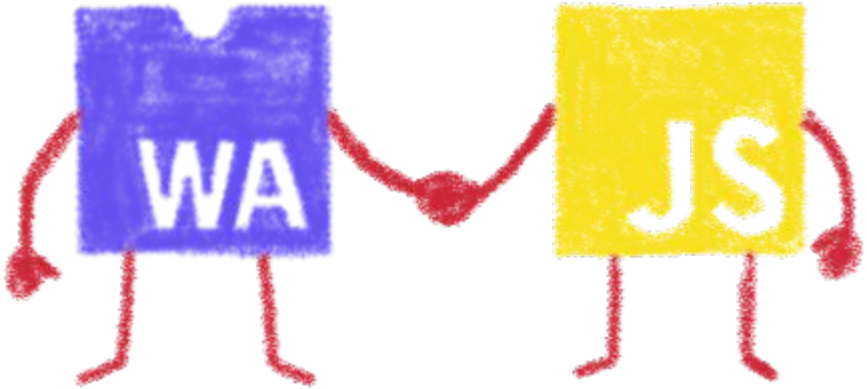
Fast & Efficient

Memory-safe & Sandboxed

Open & Deboggable

www Part of the Web Platform

But above all...



WebAssembly is not meant to replace JavaScript

Who is using WebAssembly today?

FIGMA

AUTOCAD

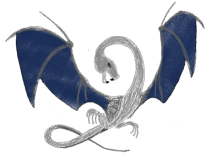
Qt

UNREAL ENGINE

unity

Google Earth

And many more others...



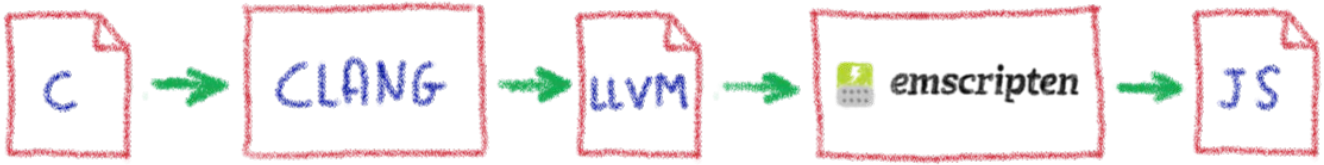
A bit of history

Remembering the past
to better understand the present

Executing other languages in the browser

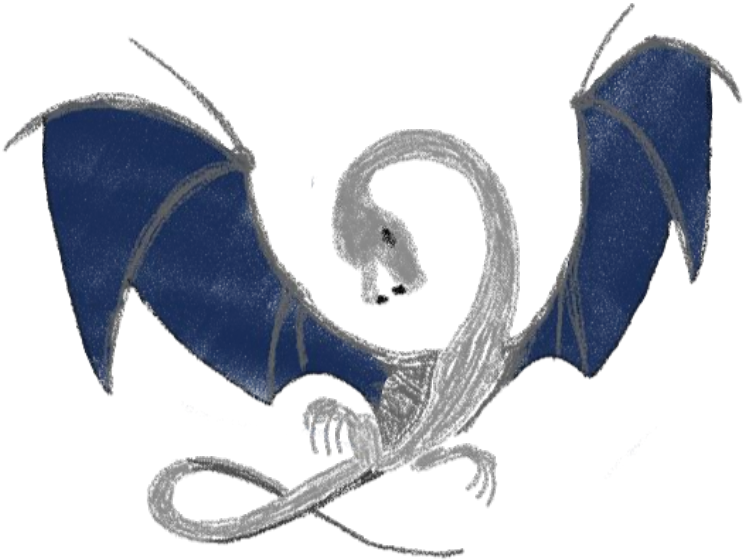
A long story, with many failures...

2012 - From C to JS: enter emscripten



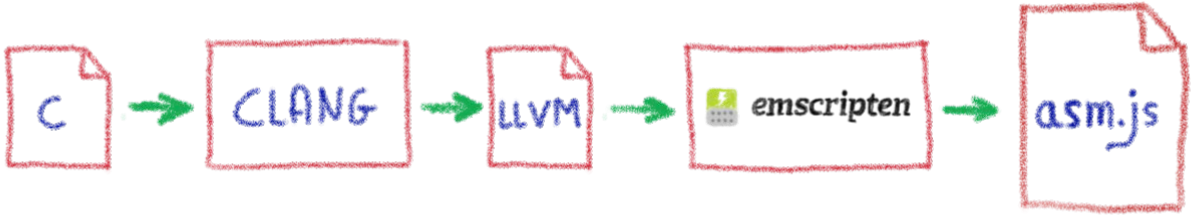
Passing by LLVM pivot

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

mozilla

Google

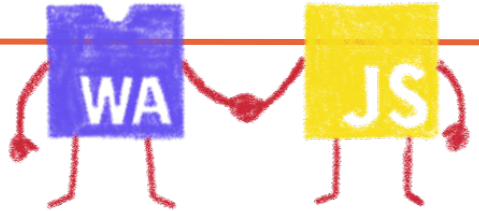
Joint effort

Microsoft

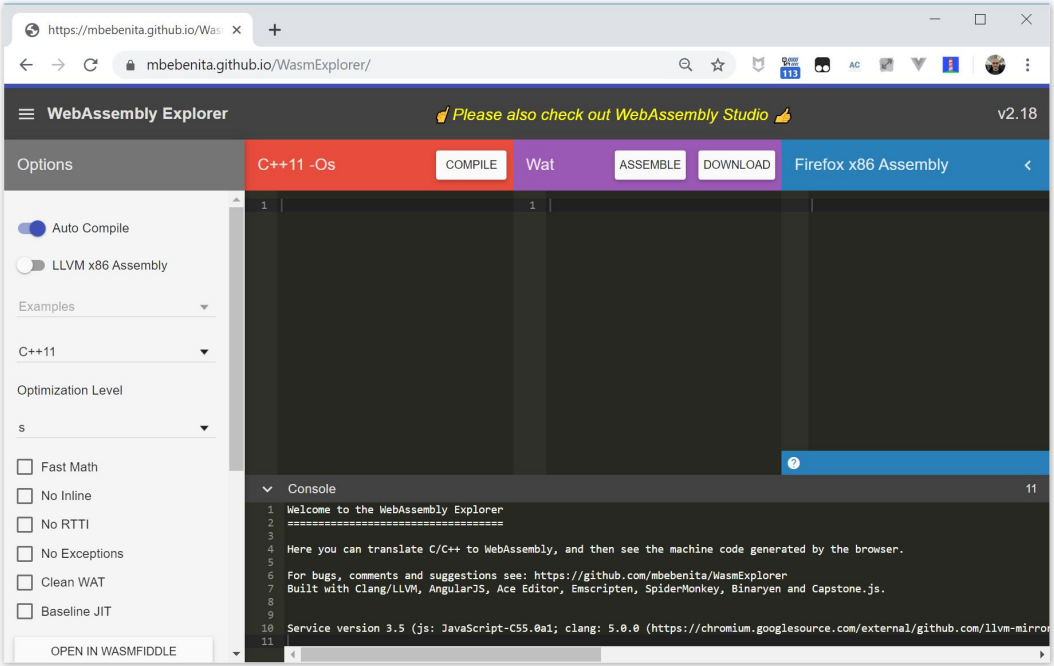
W3C

Hello W(asm)orld

My first WebAssembly program



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



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

Let's begin with the a simple function

C++11 -Os | COMPILE

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

Wat | ASSEMBLE | DOWNLOAD

```wat
(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
  )
)
```

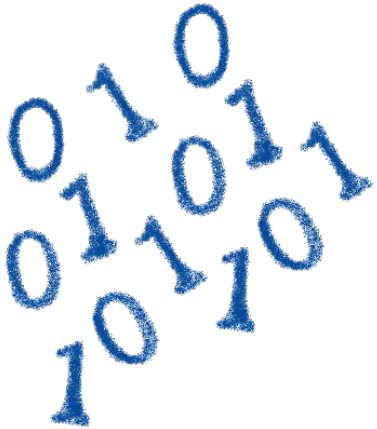
Firefox x86 Assembly

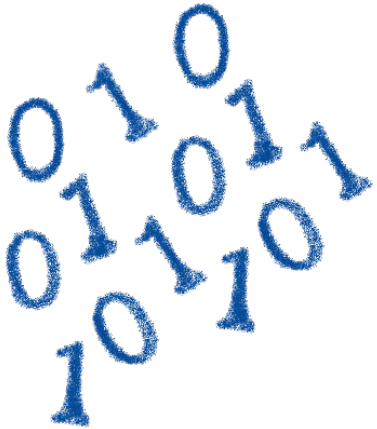
```asm
wasm-function[0]:
  sub rsp, 8
  mov edx, edi
  mov ecx, edx
  mov eax, edx
  imul ecx, eax
  mov eax, ecx
  nop
  add rsp, 8
  ret
```

WAT: WebAssembly Text Format

Human readable version of the `.wasm` binary

Download the binary `.wasm` file

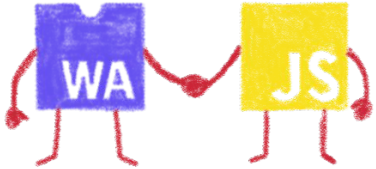




Now we need to call it from JS...

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 > ...

```javascript
var importObject = {
    imports: {
      imported_func: function(arg) {
        console.log(arg);
      }
    }
  };

async function loadWebAssembly() {
    let response = await fetch('squarer.wasm');
    let arrayBuffer = await response.arrayBuffer();
    let wasmModule = await WebAssembly.instantiate(arrayBuffer, importObject);
    squarer = await wasmModule.instance.exports._Z7squareri;
    console.log('Finished compiling! Ready when you are...');
}

loadWebAssembly();
```

Loading the `squarer` function

wasm > squarer > squarer.html > ...

```html
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <title>WASM Squarer Function</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
</head>
<body>

    <h1>WASM Squarer Function</h1>

    <script src="squarer.js"></script>

    <p>Use the browser console to calculate squares</p>
</body>
</html>
```

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

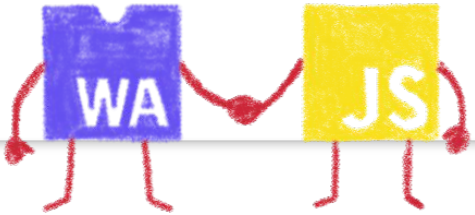
WASM Squarer Function

Use the browser console to calculate squares

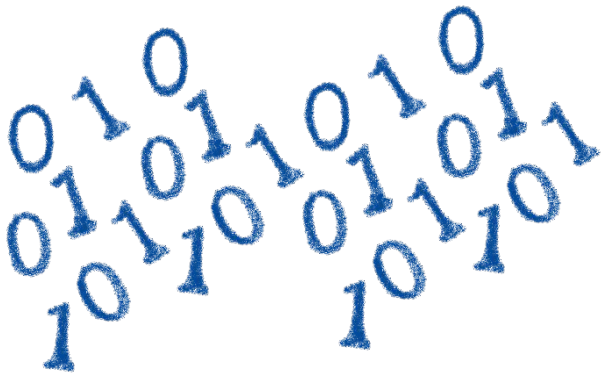
Elements | Console | Sources | Network

top | Filter | Default levels

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



Directly from the browser console
(it's a simple demo…)



You sold us a codelab!

Stop speaking and let us code

You can do steps 01 and 02 now

Let's code, mates!

Some use cases

What can I do with it?

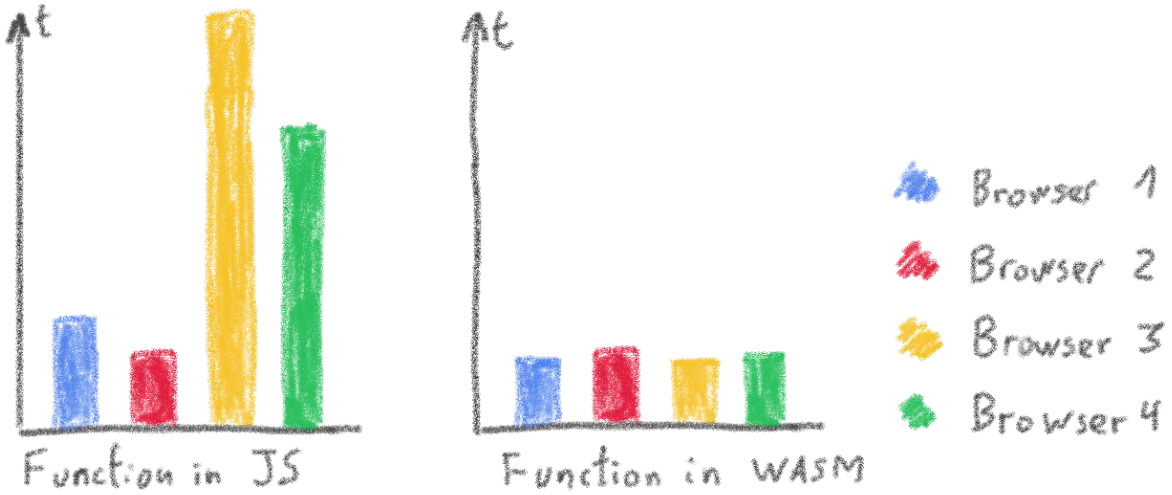
Tapping into other languages ecosystems

SQUOOSH.APP

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

Don't rewrite libs anymore

Replacing problematic JS bits

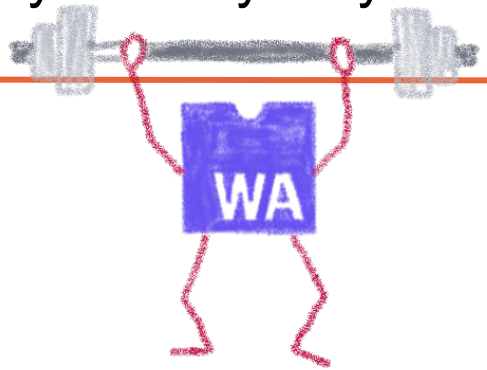


Predictable performance

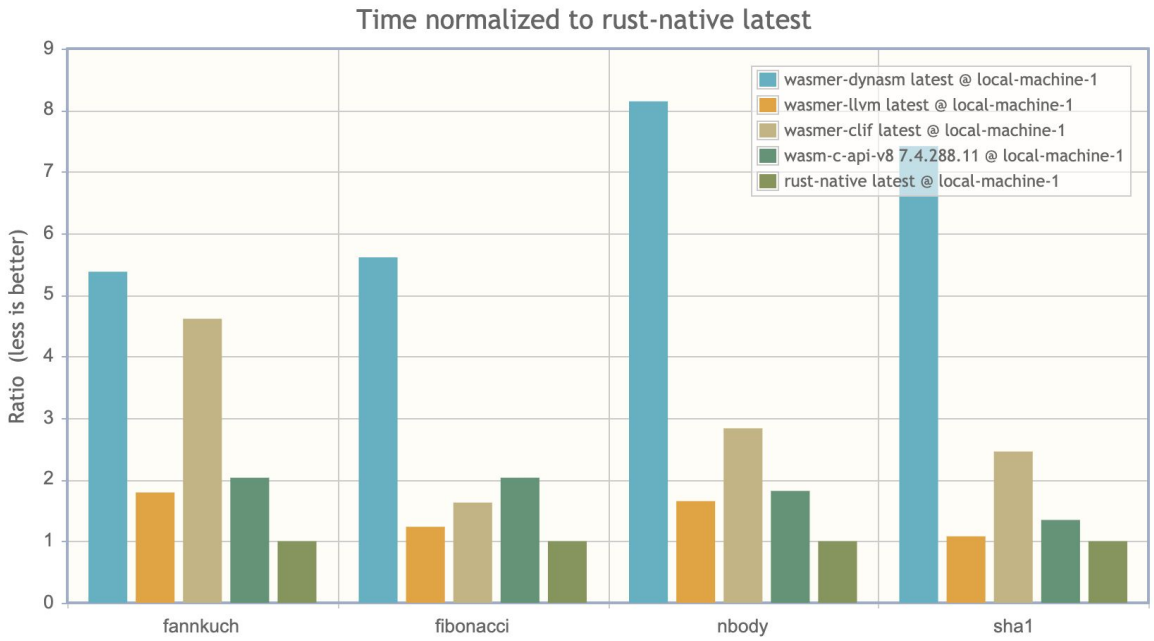
Same peak performance, but less variation

Features of Wasm

Why is everybody looking at it?



Near native speed

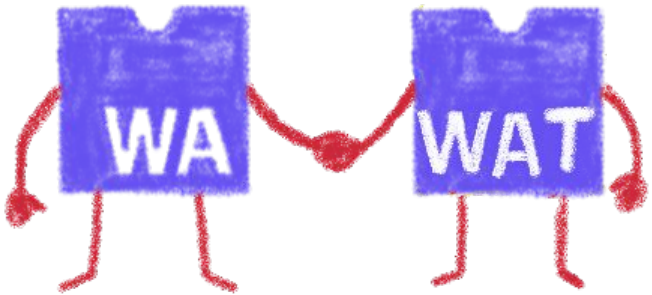


https://medium.com/wasmer/benchmarking-webassembly-runtimes-18497ce0d76e

Highly portable

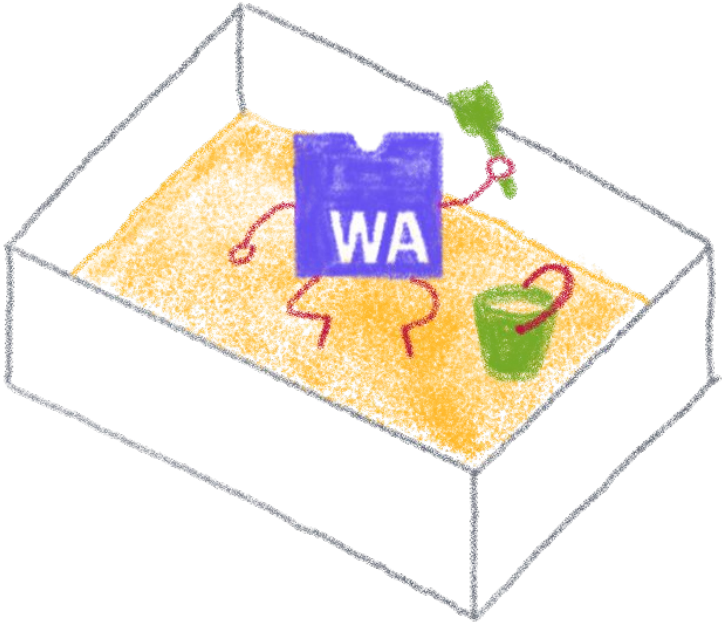
It can be run almost everywhere...

Readable and debuggable



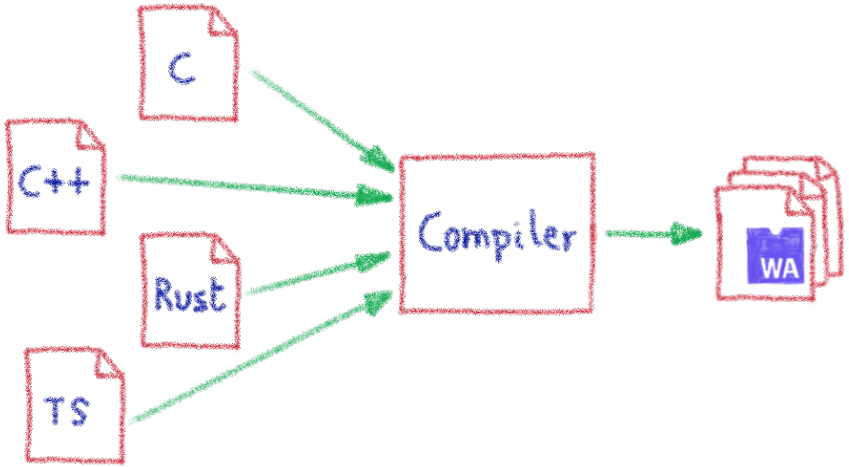
Each `.wasm` file with it `.wat` companion file

Memory safe & secure

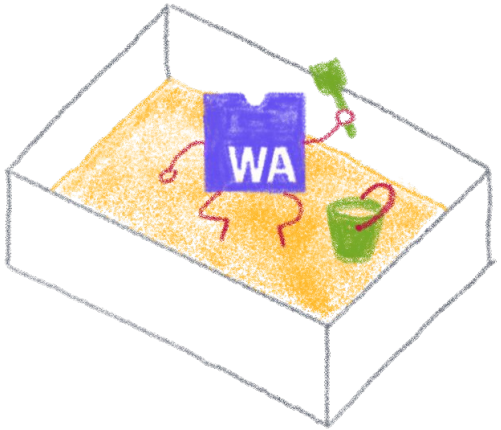


Running in a fully sandboxed environment

Accepting many source languages



And more and more...



Some constraints

Still a young platform

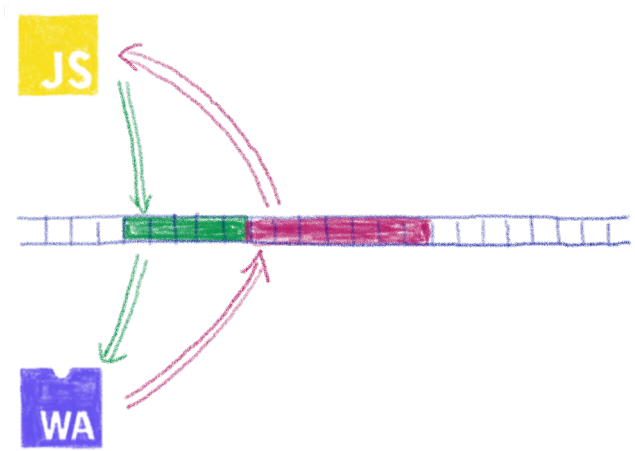
Native WASM types are limited

WASM currently has four available types:

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

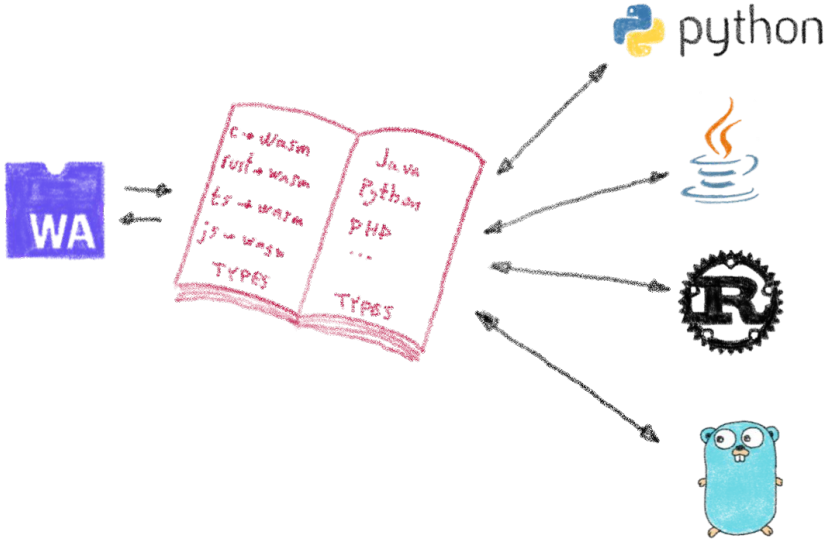
Types from languages compiled to WASM are mapped to these

How can we share data?



Using the same data in WASM and JS?
Shared linear memory between them,
and serializing the data to one Wasm types

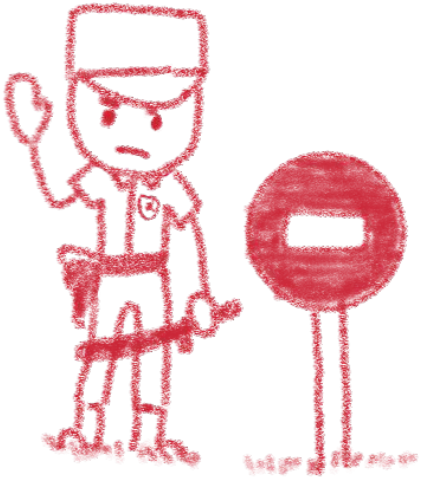
Solution is coming: Interface types



Beautiful description at:

https://hacks.mozilla.org/2019/08/webassembly-interface-types

No outside access



By design, communication is done using the shared linear memory only

Solution exists: 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.

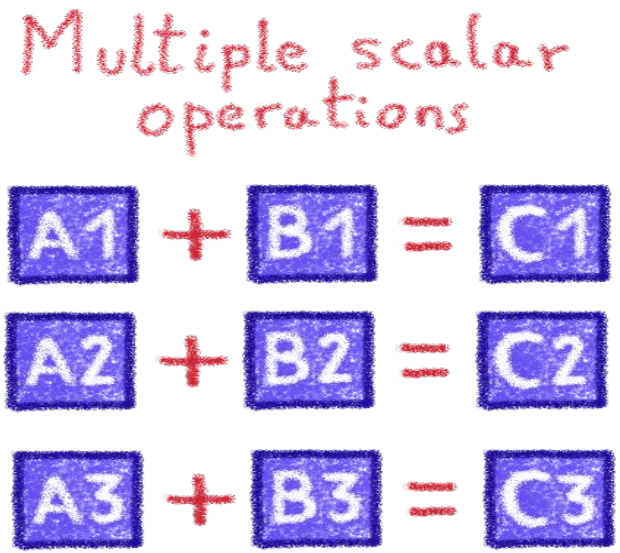
For a quick intro to WASI, including getting started using it, see the intro document.

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.

Already available in Wasmer

Mono-thread and scalar operations only



Not the most efficient way...

Solution exists: SIMD

Multiple scalar operations

A1 + B1 = C1

A2 + B2 = C2

A3 + B3 = C3

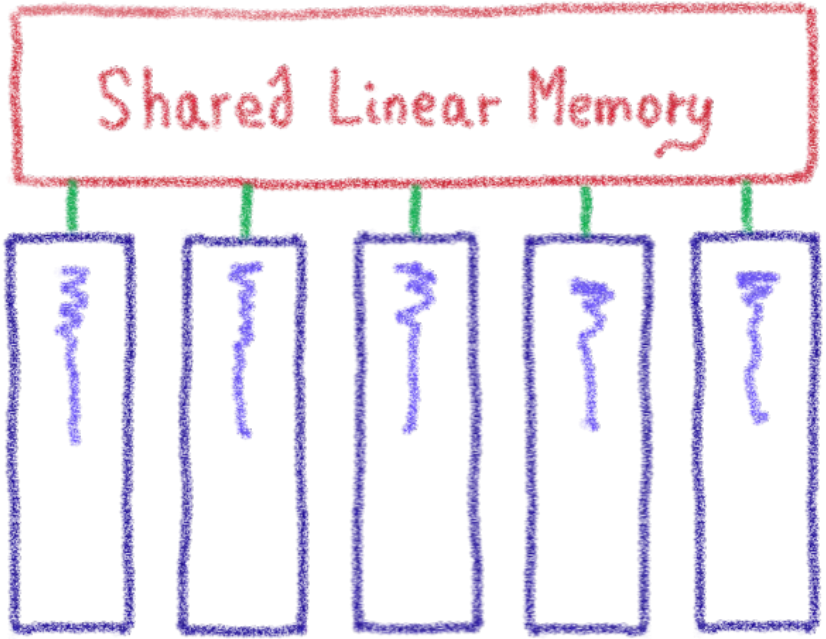
Single vectorial operation

A1		B1		C1
A2	+	B2	=	C2
A3		B3		C3

Single Instruction, Multiple Data

Already available in Wasmer

Solutions are coming too: Wasm Threads



Threads on Web Workers with shared linear memory

Incoming proposals: Garbage collector

And exception handling

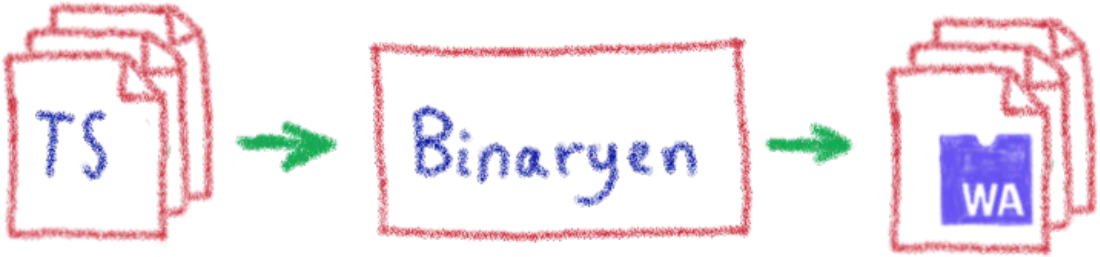
You can do steps 03 and 04 now

Let's code, mates!

AssemblyScript

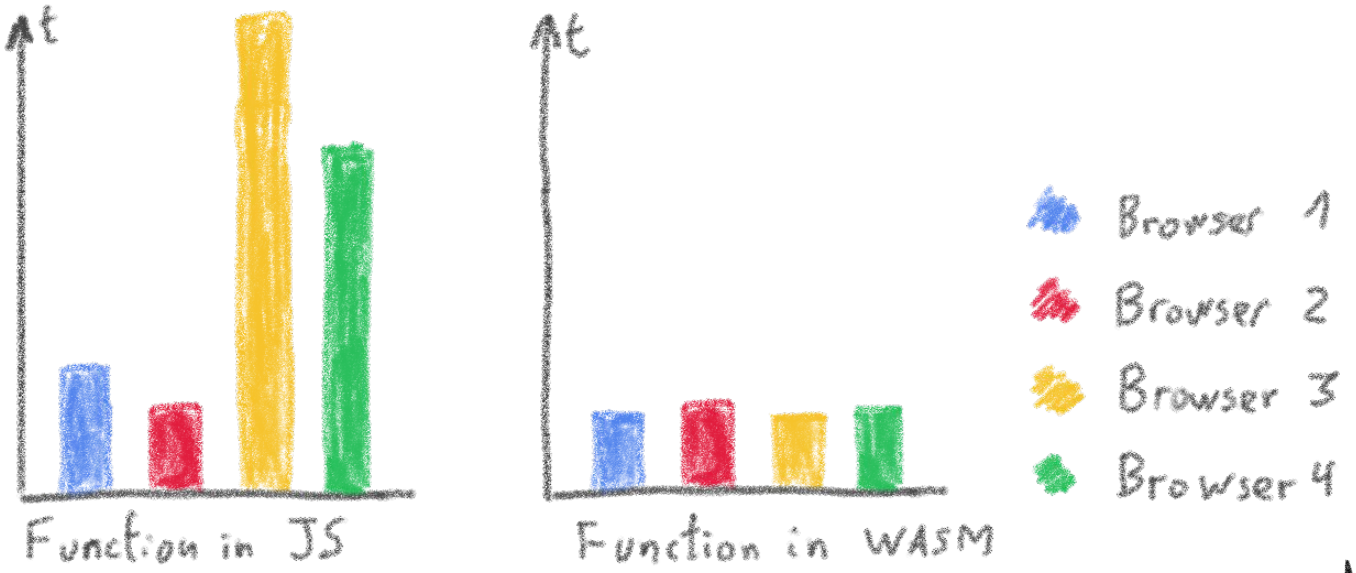
Writing WASM without learning a new language

TypeScript subset compiled to WASM



Why would I want to compile TypeScript to WASM?

Ahead of Time compiled TypeScript



More predictable performance

Avoiding the dynamicness of JavaScript

https://mbebenita.github.io/Was | WebAssembly Studio | WASM Squarer Function

webassembly.studio

BETA WebAssembly Studio

Fork | Create Gist | Download | Share | Build | Run | Build & Run | Help & Privacy | GitHub Issues

- README.md
- assembly
  - main.ts
  - tsconfig.json
  - gulpfile.js
  - package.json
  - setup.js
- src
  - main.html
  - main.js

TS main.ts

```
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

https://mbebenita.github.io/Was | WebAssembly Studio | WASM Squarer Function

webassembly.studio

BETA WebAssembly Studio

Fork | Create Gist | Download | Share | Build | Run | Build & Run | Help & Privacy | GitHub Issues

- README.md
- assembly
  - main.ts
  - tsconfig.json
  - gulpfile.js
  - package.json
  - setup.js
- src
  - main.html
  - main.js
- out
  - main.wasm
  - main.wasm.map

JS main.js

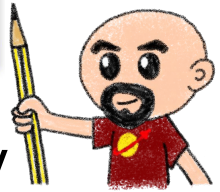
```js
WebAssembly.instantiateStreaming(fetch("../out/main.wasm"), {
  main: {
    sayHello() {
      console.log("Hello from WebAssembly!");
    }
  },
  env: {
    abort(_msg, _file, line, column) {
      console.error("abort called at main.ts:" + line + ":" + column);
    }
  },
}).then(result => {
  const exports = result.instance.exports;
  document.getElementById("container").textContent = "Result: " + exports.add(19, 23);
}).catch(console.error);
```

Output (15) | Problems (0)

16

Result: 42

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



No direct access to DOM

https://mbebenita.github.io/Was | WebAssembly Studio | WASM Squarer Function

webassembly.studio

BETA WebAssembly Studio | Fork | Create Gist | Download | Share | Build | Run | Build & Run | Help & Privacy | GitHub Issues

- README.md
- assembly
  - main.ts
  - tsconfig.json
  - gulpfile.js
  - package.json
  - setup.js
- src
  - main.html
  - main.js
- out
  - main.wasm
  - main.wasm.map

JS main.js

```js
WebAssembly.instantiateStreaming(fetch("../out/main.wasm"), {
  main: {
    sayHello() {
      console.log("Hello from WebAssembly!");
    }
  },
  env: {
    abort(_msg, _file, line, column) {
      console.error("abort called at main.ts:" + line + ":" + column);
    }
  },
}).then(result => {
  const exports = result.instance.exports;
  document.getElementById("container").textContent = "Result: " + exports.add(19, 23);
}).catch(console.error);
```

Output (15) | Problems (0)

Result: 42

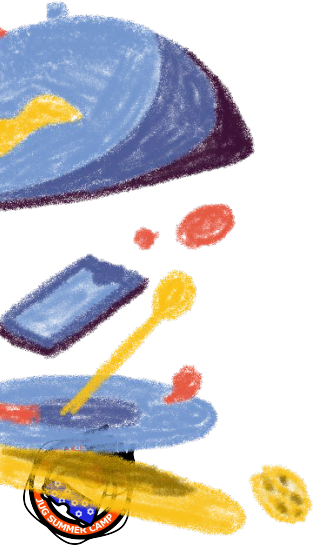
Glue code using exports/imports to/from JavaScript

You can do step 05 now

Let's code, mates!

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>
```

Template

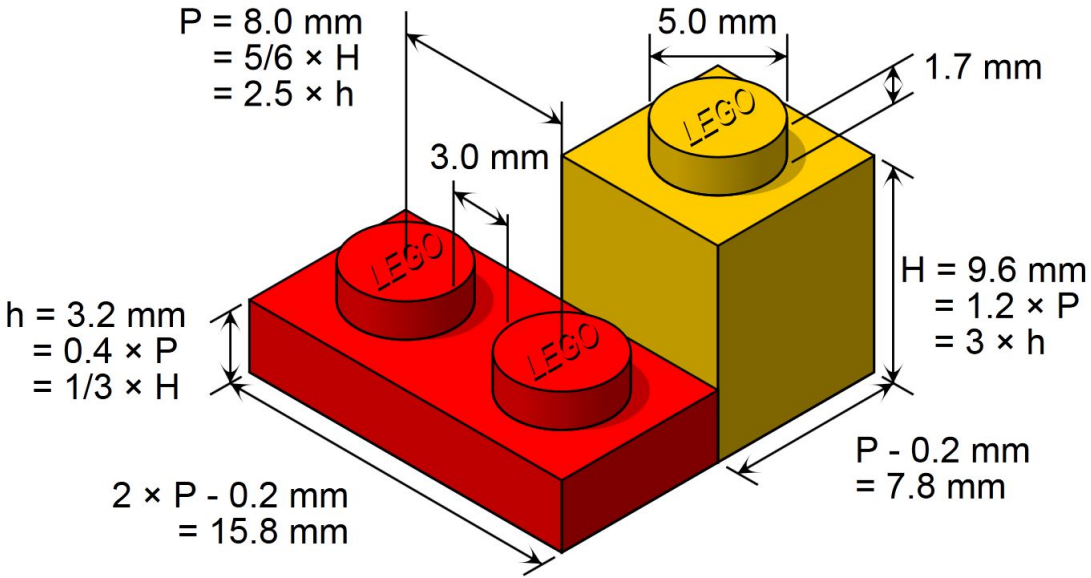
To have clonable document template

```
<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!