# Introduction to Web-Assembly

By Shaun Naude (u18014080)

## WASM Introduction:

Web-Assembly otherwise known as "WASM" is a low level language that is able to interface with JavaScript and execute within the browser. WASM was created to allow for better performance on the web by leveraging the low-level binary format of WASM. This new type of code was not meant to be written by a developer directly but rather compiled from other higher level languages such as C/C++ or C#. Thus, as a developer you will not need to understand how the intricacies of WASM works in order for you to utilize this technology. The only real learning curve associated with WASM is trying to understand how to execute your compiled WASM within JavaScript. WASM follows an open standard that is supported by the latest mainstream browsers, like JavaScript, WASM is portable and will be able to execute in any Web-Assembly compliant browser. This tutorial will attempt to demonstrate how WASM functions can be called within JavaScript to ultimately make your web applications faster in the future.

For the Purpose of this tutorial we will be building a simple calculator that will demonstrate how to utilize functions written in C within JavaScript.

## Initial Setup:

1. Firstly you will need to install **emscripten**. This is a tool that will allow for compiling to asm.js and WebAssembly thus allowing C/C++ on the web at near-native speed. For more information on the installation process follow the provided **link**.
2. Next you will need to set up a web-server, for this tutorial we will be using **python** but feel free to use an alternative. *(reason for using a web-server will be explained later)*
3. Last but not least you will need a WebAssembly compliant browser and your text editor of choice.

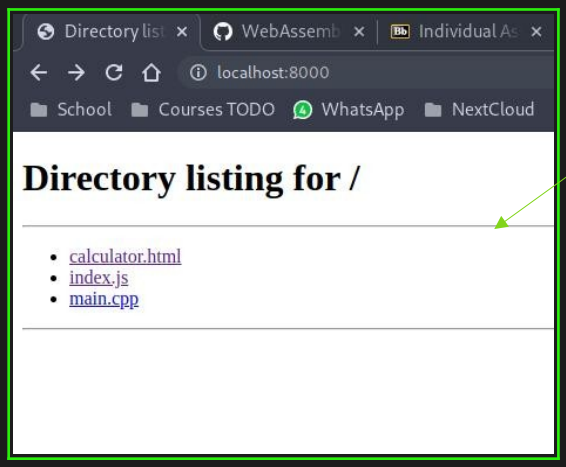
## Let's get going!:

### Step 1:

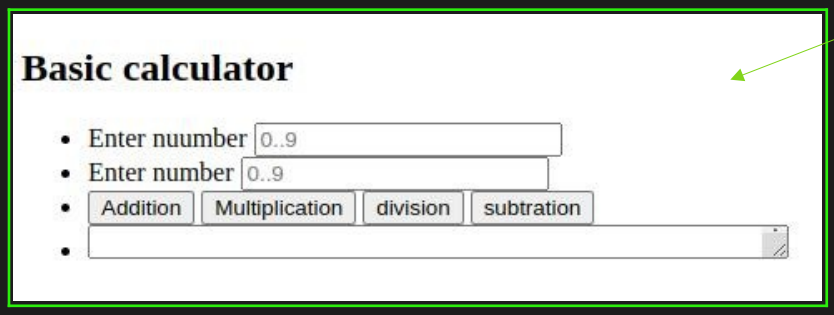
- Create a new directory with 3 files ( ***main.cpp, calculator.html and index.js)***.
- Copy the HTML from my [GitHub] and paste it in your HTML file.
- Now lets make sure you are able to serve these files. Open a terminal within the directory containing your code and execute the following command within the terminal.

```
shaun@Home-PC:~/Code/WebAssembly-Tutorial/Code/src|master
⇒  python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

Once this has been done open your browser that the specified address.




You should be able to see all the files being hosted within the directory where you started the server.




If you click on the calculator.html file this is what should render if you have copied the HTML correctly.

### Reasoning for Using web-Server:

You might be wandering why we have a python web-server hosting our files instead of just opening the plain HTML file within a browser ? The main reason for hosting the files is because they will contain asynchronous requests and most modern browser will not allow this to take place on a local file due to security reasons.

---

## Step 2:

Now that we have the initial setup completed we can start writing the C functions that will allow your basic calculator to work.

- Firstly open the ***main.cpp*** in your text editor of choice.

When writing your C code remember to wrap your functions with (extern “C”). The Reason for this is to avoid **C++ name mangling.**

Now that you have added the the (extern “C”) wrapping you are able to freely write C functions that we will later use within our JavaScript.

main.cpp

src > main.cpp > ...

```
1   extern "C" {
2   #include <math.h>
3   float add(float x,float y) {
4     return (x+y);
5   }
6
7   float multiply(float x,float y){
8     return (x*y);
9   }
10
11  float divide(float x,float y){
12    return (x/y);
13  }
14
15  float subtract(float x, float y){
16    return (x-y);
17  }
18
19  }
```

- Now that we have written our C functions lets compile this code into Web-Assembly so that we will be able to link it via JavaScript to our HTML page. We will be Using the emscripten tool that we installed earlier.

As shown add the name of your cpp file.

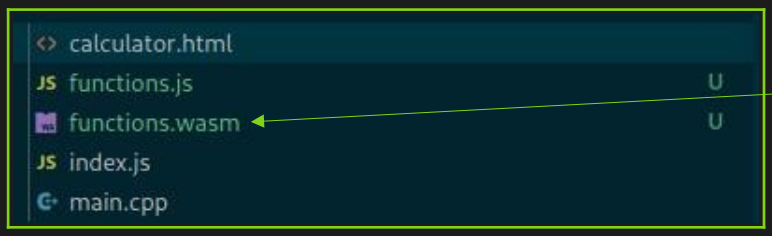
```
shaun@Home-PC:~/Code/WebAssembly-Tutorial/Code/src master
→  emcc main.cpp -o functions.js -s EXPORTED_FUNCTIONS='["_add","_multiply","_divide","_subtract"]' -s EXPORTED_RUNTIME_METHODS='["ccall", "cwrap"]'
```

Give the *.js* file that is generated a name this file will need to be imported within your HTML to allow your self coded JavaScript to interact with the WebAssembly.

Here we will define the functions that we want to export from the given *.cpp* file. Note that each function name must be preceded by "_".

Here we are exporting the methods that will allow our JavaScript to interact with the Web Assembly. These methods will make more sense once we start writing our own JavaScript.

- Now your directory where your above code was written should look similar to the image below.




There is our actual WebAssembly file.

---

## Step 3:

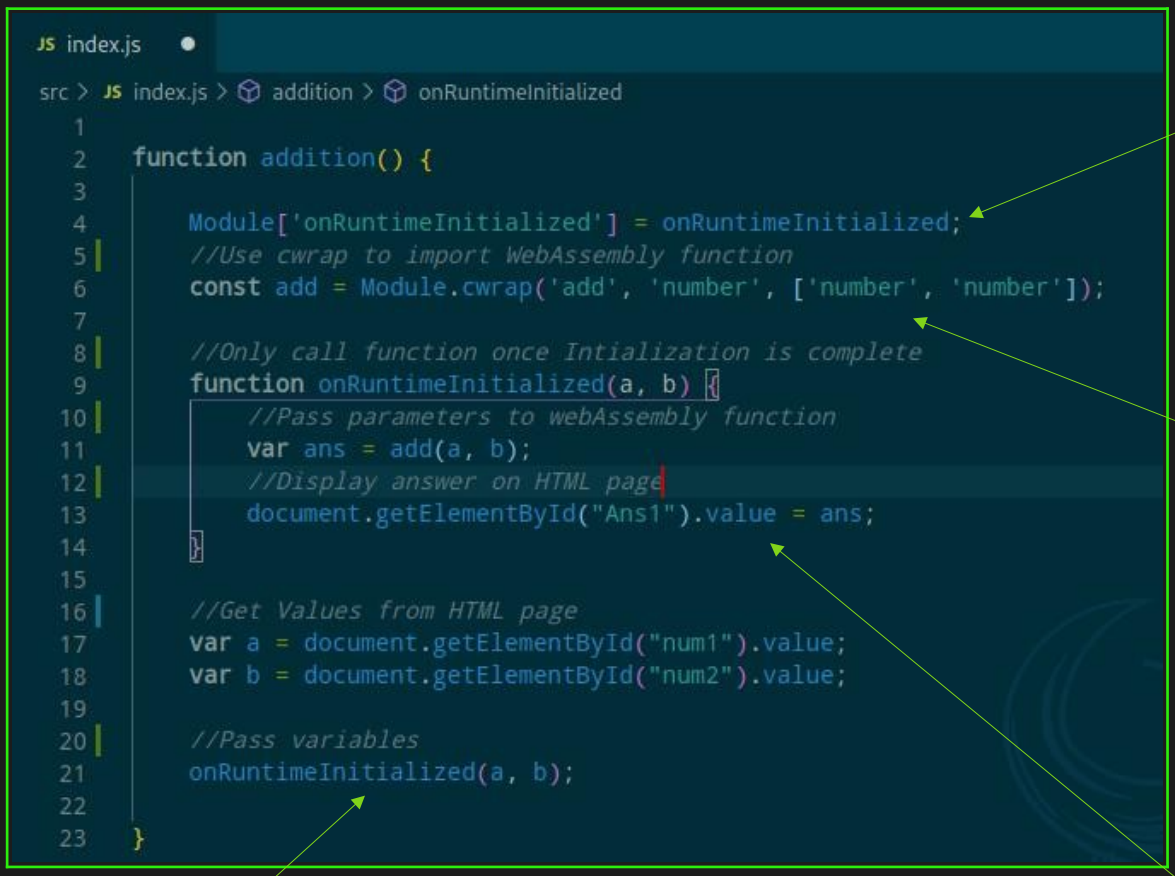
Now last but not least we will be calling our Web-Assembly functions within our ***index.js*** to allow our basic-calculator to function.

- Firstly ensure that both the generated JavaScript file from the emscripten tool as well as your own JavaScript file *(mine is called index.js)* has been added to your HTML file.

```
<script src="functions.js"></script>
<script src="index.js"></script>
```

If you by any chance did not use the same naming convention as what I did ensure that you have correctly added both JavaScript files to your HTML page.

- Now lets open index.js and begin using our web-Assembly.




We start by creating a function that can only be executed once the HTML page has been initialised. This is needed because we need to wait until it is safe to call compiled functions.

Here we import the **add** function using cwrap. The first parameter is the **name** of the function we would like to import. The second parameter defines the **data-type** that the function will return and lastly the third parameter defines the **input parameters**.

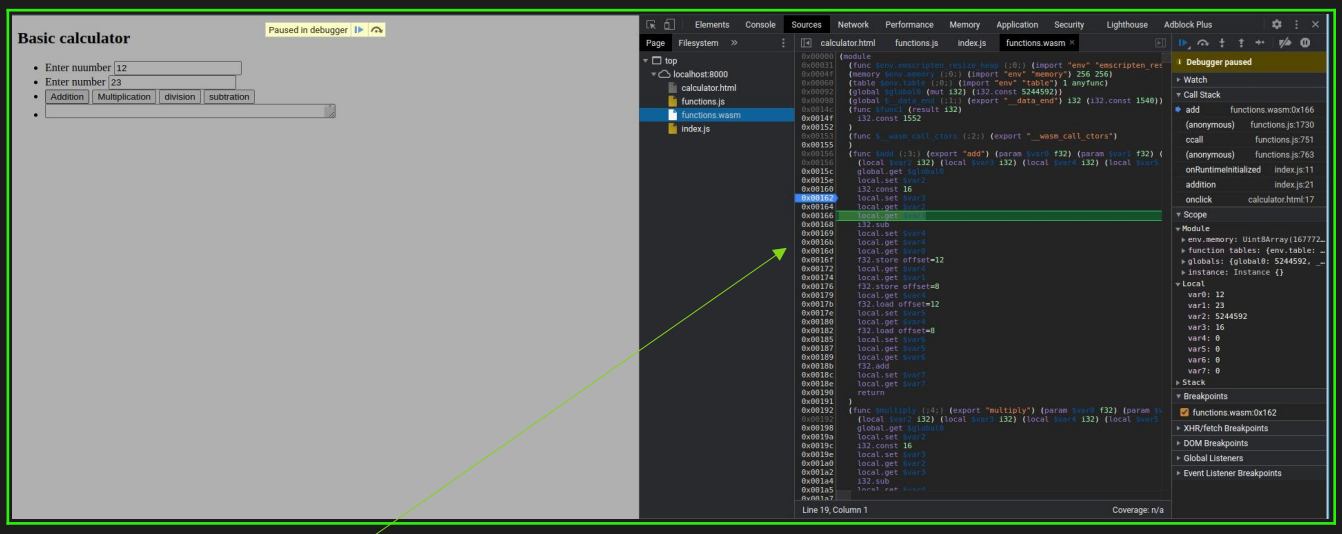
Here we call our intermediate function that will call our actual WebAssembly function.

- Note that for all the other functions such as multiply, divide or subtract they will operate in the exact same fashion with only the function being imported in line 6 being the differentiating factor.

# Extra Information:

## Viewing the WebAssembly

If you would like to see the WebAssembly in action you can set a break point within the ***.wasm*** file and view the code execute line by line.




In the above example the addition button was pressed thus our JavaScript called the add() function from the .wasm file. This proves that you are infact using the wasm as intended.

## Full Example

If you would like to view the full source code from this tutorial please visit my [GitHub]

# References:

(2020) *Compiling a New C/C++ Module to WebAssembly,*Available at:*https://developer.mozilla.org/en-US/docs/WebAssembly/*(Accessed: 2020).

(2020) *How do you set up a local testing server?,* Available at: *https://developer.mozilla.org/en-US/docs/WebAssembly/* (Accessed: 2020).

(2020) *Name_mangling,* Available at: *https://en.wikipedia.org/wiki/Name_mangling* (Accessed: 2020)

(2020) *Interacting with code,*Available at:*https://emscripten.org/docs/porting/connecting_cpp_and_javascript/Interacting-with-code.html#calling-compiled-c-functions-from-javascript-using-ccall-cwrap* (Accessed: 2020).