

# Lab1

Yiwen Xu

48377645

## Setup the Emscripten

I completed the installation by following the instructions in the [official documentation](#).

### Download and install using emsdk

```
# Get the emsdk repo
git clone https://github.com/emscripten-core/emsdk.git
```

```
[(base) evelynxu@jos-MacBook-Pro-2 ~ % git clone https://github.com/emscripten-core/emsdk.git
Cloning into 'emsdk'...
remote: Enumerating objects: 3385, done.
remote: Counting objects: 100% (1/1), done.
remote: Total 3385 (delta 0), reused 0 (delta 0), pack-reused 3384
Receiving objects: 100% (3385/3385), 1.95 MiB | 468.00 KiB/s, done.
Resolving deltas: 100% (2222/2222), done.
[(base) evelynxu@jos-MacBook-Pro-2 ~ % cmake --version
cmake version 3.22.2

CMake suite maintained and supported by Kitware (kitware.com/cmake).
```

```
# Fetch the latest version of the emsdk (not needed the first time)
git pull

# Download and install the latest SDK tools.
# Will installed nodejs, python, etc.
./emsdk install latest

# Make the "latest" SDK "active" for the current user. (writes .emscripten file)
./emsdk activate latest

# Activate PATH and other environment variables in the current terminal
source ./emsdk_env.sh
```

- Install

```
(base) evelynxu@jos-MacBook-Pro-2 emsdk % ./emsdk install latest
Resolving SDK alias 'latest' to '3.1.20'
Resolving SDK version '3.1.20' to 'sdk-releases-upstream-d92c8639f406582d70a5dde27855f74ecf602f45-64bit'
Installing SDK 'sdk-releases-upstream-d92c8639f406582d70a5dde27855f74ecf602f45-64bit'..
Installing tool 'node-14.18.2-64bit'..
Downloading: /Users/evelynxu/emsdk/zips/node-v14.18.2-darwin-x64.tar.gz from https://storage.googleapis.com/webassembly/emscripten-releases-builds/deps/node-v14.18.2-darwin-x64.tar.gz, 32076686 Bytes
Unpacking '/Users/evelynxu/emsdk/zips/node-v14.18.2-darwin-x64.tar.gz' to '/Users/evelynxu/emsdk/node/14.18.2_64bit'
Done installing tool 'node-14.18.2-64bit'.
Installing tool 'python-3.9.2-64bit'..
Downloading: /Users/evelynxu/emsdk/zips/python-3.9.2-3-macos-x86_64.tar.gz from https://storage.googleapis.com/webassembly/emscripten-releases-builds/deps/python-3.9.2-3-macos-x86_64.tar.gz, 31899321 Bytes
Unpacking '/Users/evelynxu/emsdk/zips/python-3.9.2-3-macos-x86_64.tar.gz' to '/Users/evelynxu/emsdk/python/3.9.2_64bit'
Done installing tool 'python-3.9.2-64bit'.
Installing tool 'releases-upstream-d92c8639f406582d70a5dde27855f74ecf602f45-64bit'..
Downloading: /Users/evelynxu/emsdk/zips/d92c8639f406582d70a5dde27855f74ecf602f45-wasm-binaries.tbz2 from https://storage.googleapis.com/webassembly/emscripten-releases-builds/mac/d92c8639f406582d70a5dde27855f74ecf602f45/wasm-binaries.tbz2, 349627729 Bytes
Unpacking '/Users/evelynxu/emsdk/zips/d92c8639f406582d70a5dde27855f74ecf602f45-wasm-binaries.tbz2' to '/Users/evelynxu/emsdk/upstream'
Done installing tool 'releases-upstream-d92c8639f406582d70a5dde27855f74ecf602f45-64bit'.
Done installing SDK 'sdk-releases-upstream-d92c8639f406582d70a5dde27855f74ecf602f45-64bit'.
```

- Activate SDK

```
[(base) evelynxu@jos-MacBook-Pro-2 emsdk % ./emsdk activate latest
Resolving SDK alias 'latest' to '3.1.20'
Resolving SDK version '3.1.20' to 'sdk-releases-upstream-d92c8639f406582d70a5dde27855f74ecf602f45-64bit'
Setting the following tools as active:
    node-14.18.2-64bit
    python-3.9.2-64bit
    releases-upstream-d92c8639f406582d70a5dde27855f74ecf602f45-64bit

Next steps:
- To conveniently access emsdk tools from the command line,
  consider adding the following directories to your PATH:
    /Users/evelynxu/emsdk
    /Users/evelynxu/emsdk/node/14.18.2_64bit/bin
    /Users/evelynxu/emsdk/upstream/emscripten
- This can be done for the current shell by running:
    source "/Users/evelynxu/emsdk/emsdk_env.sh"
- Configure emsdk in your shell startup scripts by running:
    echo 'source "/Users/evelynxu/emsdk/emsdk_env.sh"' >> $HOME/.zprofile
```

- Activate PATH

```

(base) evelynxu@jos-MacBook-Pro-2 emsdk % source ./emsdk_env.sh
Setting up EMSDK environment (suppress these messages with EMSDK_QUIET=1)
Adding directories to PATH:
PATH += /Users/evelynxu/emsdk
PATH += /Users/evelynxu/emsdk/upstream/emscripten
PATH += /Users/evelynxu/emsdk/node/14.18.2_64bit/bin

Setting environment variables:
PATH = /Users/evelynxu/emsdk:/Users/evelynxu/emsdk/upstream/emscripten:/Users/evelynxu/emsdk/node/14.18.2_64bit/bin:/Users/evelynxu/opt/anaconda3/bin:/Users/evelynxu/opt/anaconda3/condabin:/Library/Frameworks/Python.framework/Versions/3.9/bin:/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/opt/X11/bin:/Library/Apple/usr/bin
EMSDK = /Users/evelynxu/emsdk
EM_CONFIG = /Users/evelynxu/emsdk/.emscripten
EMSDK_NODE = /Users/evelynxu/emsdk/node/14.18.2_64bit/bin/node
EMSDK_PYTHON = /Users/evelynxu/emsdk/python/3.9.2_64bit/bin/python3
SSL_CERT_FILE = /Users/evelynxu/emsdk/python/3.9.2_64bit/lib/python3.9/site-packages/certifi/cacert.pem

```

## Verifying the installation

```
emcc --version
```

- Verify

```

(base) evelynxu@jos-MacBook-Pro-2 emsdk % emcc --version
emcc (Emscripten gcc/clang-like replacement + linker emulating GNU ld) 3.1.20 (5d878c99921ec247d34fb26a20b5a13d60d69e93)
Copyright (C) 2014 the Emscripten authors (see AUTHORS.txt)
This is free and open source software under the MIT license.
There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

```

## Running Emscripten

We can now compile C/C++ file to JavaScript.

Note: Our newly opened terminal window is not capable of executing `emcc` and needs to be activated each time with the following command.

```

cd emsdk
./emsdk activate latest
source ./emsdk_env.sh

```

First, we create a C++ file: `hello_world.cpp`.

```
#include <iostream>

using namespace std;

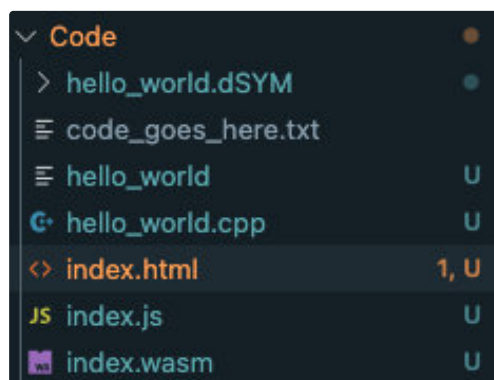
int main(int argc, const char * argv[]){
    cout << "hello world\n";
    return 0;
}
```

To build the JavaScript version of this code. We open a new terminal, active `emcc`. Go from this terminal window to the folder where you saved this C++ file. Then execute the following command in the terminal.

```
emcc hello_world.cpp -o index.html
```

After that, we get three files:

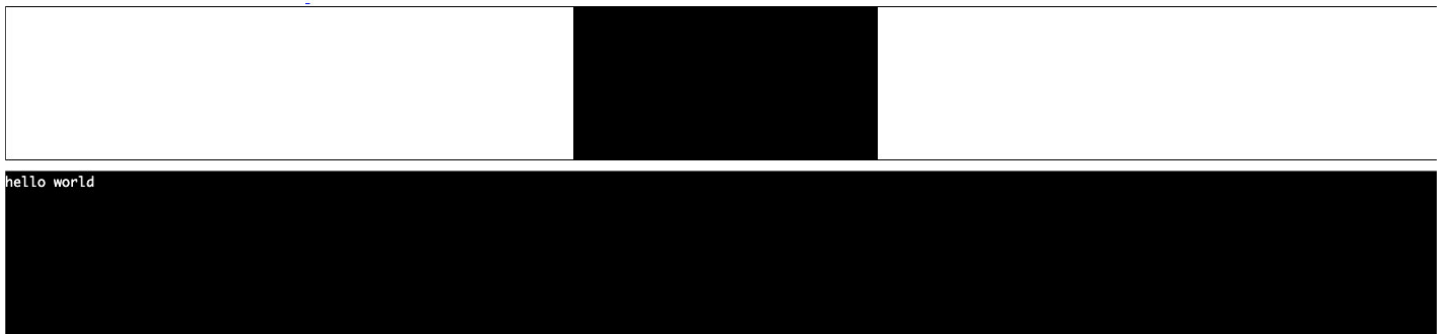
- index.html
- index.js
- index.wasm



Open the HTML file through Live Sever. We can see `hello world` shows correctly on this page.



☐ Resize canvas ☒ Lock/hide mouse pointer



## Identify code base

I wrote a Fibonacci output function based on C++ language as a code base.

```
fib.cpp — lab-1-emscripten-EvelynXu22
Code > fib.cpp > main(int, const char * [])
1  #include <iostream>
2
3  #define N 40    // Output N Fibonacci numbers
4
5  using namespace std;
6
7  int Fib(int i){    // Recursive Fibonacci number
8      if( i < 2) return i;  int Fib(int i)
9      return Fib(i - 1) + Fib(i - 2);
10 }
11
12 int main(int argc, const char * argv[]){
13     cout << "Display the Fibonacci sequence recursively:" << endl;
14     for (int i = 1; i < N; i++){
15         cout.width(10);
16         cout << Fib(i);
17     }
18     return 0;
19 }
20
```

OUTPUT    TERMINAL    JUPYTER    DEBUG CONSOLE    Filter (e.g. text, !exclude)

```
Loaded '/usr/lib/libc++abi.dylib'. Symbols loaded.
Loaded '/usr/lib/libobjc.A.dylib'. Symbols loaded.
Loaded '/usr/lib/liboah.dylib'. Symbols loaded.
Loaded '/usr/lib/libc++.1.dylib'. Symbols loaded.
Display the Fibonacci sequence recursively:
    1         1         2         3         5         8        13        21        34        55
89   144       233       377       610       987       1597       2584       4181       6765      10946
17711   28657   46368   75025  121393  196418  317811  514229  832040  1346269  21783
09  3524578  5702887  9227465 14930352 24157817 39088169 63245986
The program '/Users/evelynxu/github-classroom/CS-7345-Adv-App-Programming/lab-1-emscripten-EvelynXu22/C
ode/fib' has exited with code 0 (0x00000000).
```

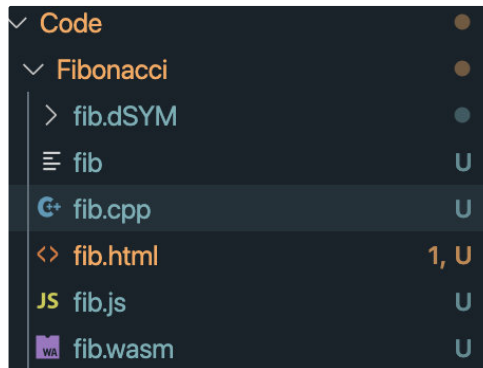
> Please start a debug session to evaluate expressions


## Test code base compiling

Then I used the same way and commands before to verify code base is functional and to ensure compiling is possible.


```
emcc fib.cpp -o fib.html
```

I get three new files which can run through Live Server.



 **powered by**  
**emscripten**

☐ Resize canvas ☒ Lock/hide mouse pointer Fullscreen



Display the Fibonacci sequence recursively:  
89      1      1      2      3      5      8      13      21      34      55  
144    233    377    610    987    1597    2584    4181

## Build code base to WebAssembly

Deleted those three transpiling files before. We now only compile WebAssembly file and JavaScript file with `emcc`.

First, I changed part of the code base to prepare a interface. And return 1 to prove that all work is done.

```
#include <iostream>

#define N 30 // Output N Fibonacci numbers

using namespace std;

int fibArray[N];

int fib(int i)
{ // Recursively get Fibonacci number
  if (i < 2)
    return i == 1;
  return fib(i - 1) + fib(i - 2);
}

extern "C"
{
  int getFib()
  {
    // static int fibArray[N];
    for (int i = 1; i < N; i++)
    {
      fibArray[i] = fib(i);
    }
    return 1;
  }
}
```

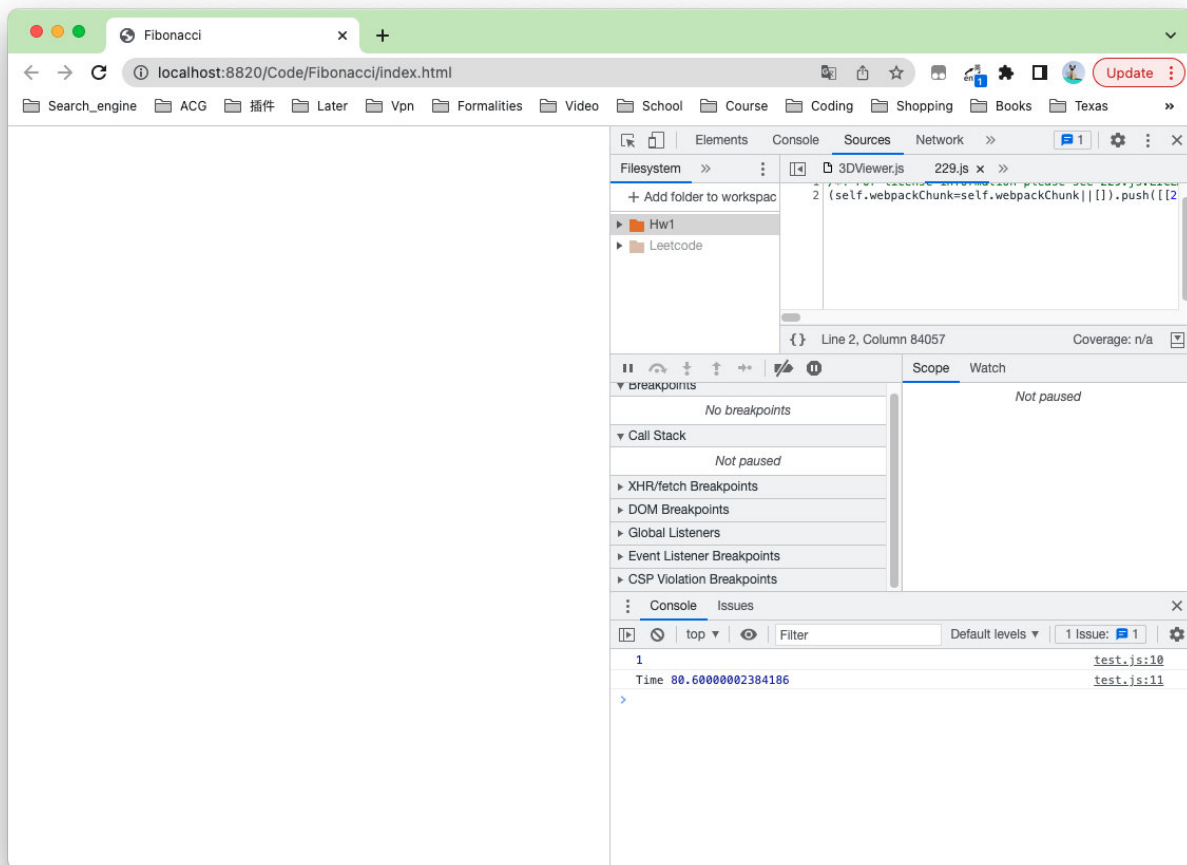
Then, compile it with this command below:

```
Fibonacci % emcc fib.cpp -o fib.js -s EXPORTED_FUNCTIONS=_getFib -s
EXPORTED_RUNTIME_METHODS=ccall
```

Create an a `test.js` and `index.html` file to call `getFib` method and show the execution time on console.

```
Code > Fibonacci > JS test.js > ...
1
2
3 // var Module = require("fib.js");
4 var test = Module.onRuntimeInitialized = () => {
5   // for (var i = 0; i < 100; i++){
6     var t0 = performance.now();
7     var res = Module.ccall('getFib', 'number');
8     var t1 = performance.now();
9     asmTime = t1 - t0;
10    console.log(res);
11    console.log("Time", asmTime);
```





## Compare execution time

In this step, I ran this code base 100 times both in C++ and in browser and save the results in `cpp_time.txt` and `browser_time.txt`.



Fibonacci

localhost:8820/Code/Fibonacci/index.html

Search\_engine ACG 插件 Later Vpn Formalities Video School Course Coding Shopping Books Texas

1.007700000476837  
0.926200000476838  
0.862899999761582  
0.8455  
0.838300000715256  
0.836899999761582  
0.8355  
0.832799999523162  
0.833899999761582  
0.828100000238419  
0.886100000238418  
0.843799999523162  
0.840299999523163  
0.850200000476837  
0.8385  
0.977399999761581  
0.875300000715255  
0.844700000476838  
0.845299999523163  
0.839600000238418  
0.861799999523163  
0.864699999284745  
0.911  
0.837700000476837  
0.841799999523162  
0.871699999284745  
0.935600000238418  
0.855899999761582  
0.900899999761581  
0.866300000715255  
0.942600000238418  
0.867199999284744  
0.837399999761581  
0.843800000715256  
0.844800000715256  
0.8775  
0.957100000238419  
0.871900000953674  
0.884100000238418

Elements Console Sources Network

3DViewer.js

```
84 var msg = "Shader program failed to link.  
85 + "<pre>" + gl.getProgramInfoLog( prog  
86 alert( msg );  
87 return -1;  
88 }  
89 }  
90 }  
91 }  
92 }
```

Scope Watch

breakpoints

No breakpoints

Call Stack

Not paused

XHR/fetch Breakpoints

DOM Breakpoints

Global Listeners

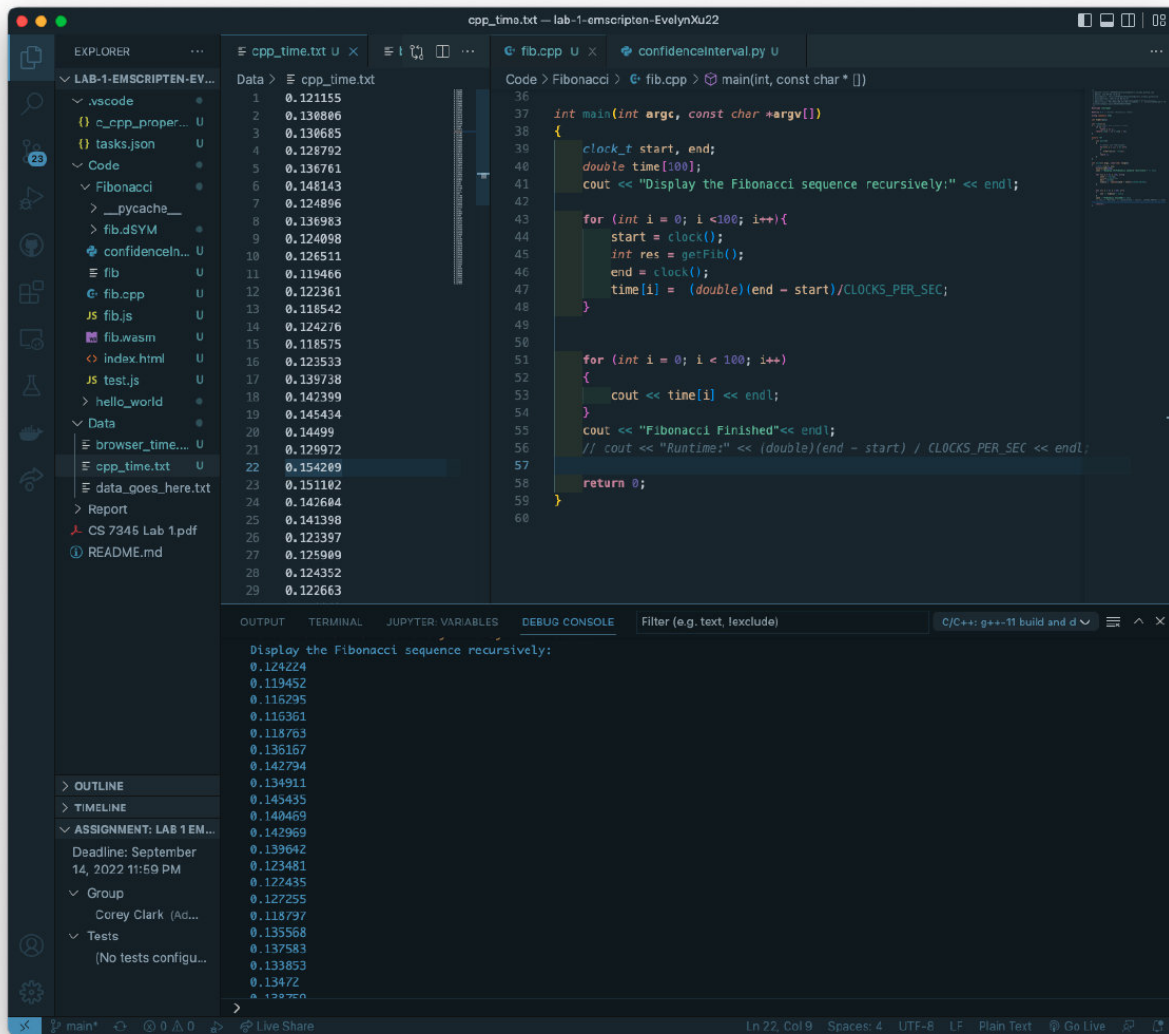
Event Listener Breakpoints

CSP Violation Breakpoints

Console Issues

2 Issues: 1 1

Time 1.007700000476837	test.js:11
Time 0.926200000476838	test.js:11
Time 0.862899999761582	test.js:11
Time 0.8455	test.js:11
Time 0.838300000715256	test.js:11
Time 0.836899999761582	test.js:11
Time 0.8355	test.js:11
Time 0.832799999523162	test.js:11
Time 0.833899999761582	test.js:11
Time 0.828100000238419	test.js:11
Time 0.886100000238418	test.js:11



Next, I wrote a python-based code for running time analysis and confidence interval calculation, named `confidenceInterval.py`. This function read the results were saved before and calculate confidenceInterval with `numpy` and `scipy`.

```
confidenceInterval.py — lab-1-emscripten-EvelynXu22

EXPLORER
LAB-1-EMSCRIPTEN-EV...
  .vscode
  {} c_cpp_proper... U
  {} tasks.json U
  Code
  Fibonacci
  > __pycache__
  > fib.dSYM
  confidenceInterval.py U
  fib
  fib.cpp U
  fib.js U
  fib.wasm U
  index.html U
  test.js U
  > hello_world
  > Data
  browser_time.txt U
  cpp_time.txt U
  data_goes_here.txt
  > Report
  CS 7345 Lab 1.pdf
  > OUTLINE
  > TIMELINE
  > ASSIGNMENT: LAB 1 EM...
  Deadline: September 14, 2022 11:59 PM
  > Group
  Corey Clark (Ad...
  > Tests

Code > Fibonacci > confidenceInterval.py > ...
1 import numpy as np
2 from scipy import stats
3
4 def getConfidenceInterval(array):
5     dt = np.dtype(np.float64)
6     array = np.array(array, dtype=dt)
7     mean, std = array.mean(), array.std(ddof = 1)
8     conf_intveral = stats.norm.interval(0.95, loc=mean, scale=std)
9     print("Confidence Interval:", conf_intveral)
10
11
12 browserTime, cppTime = [], []
13
14 path = "Data/browser_time.txt"
15 with open(path, 'r') as f:
16     browserTime = f.read().splitlines()
17
18 path = "Data/cpp_time.txt"
19 with open(path, 'r') as f:
20     cppTime = f.read().splitlines()
21
22 getConfidenceInterval(browserTime)
23 getConfidenceInterval(cppTime)
24
25 # cppTime = np.array(cppTime, dtype=dt)
26 # print(cppTime)
27

OUTPUT TERMINAL JUPYTER: VARIABLES DEBUG CONSOLE
Confidence Interval: (0.5419682017893873, 1.228425798206798)
Confidence Interval: (0.10579144538509686, 0.14679045461490317)
(base) evelynxu@os-MacBook-Pro-2 lab-1-emscripten-EvelynXu22 %
```

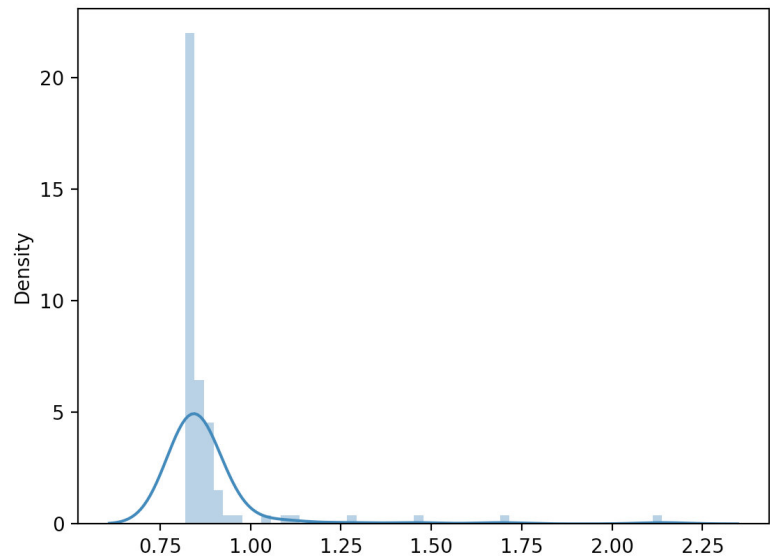
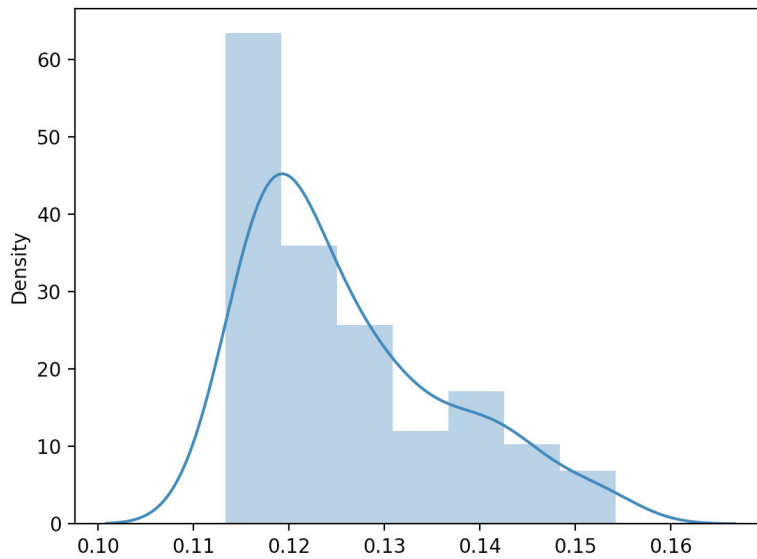
As the screenshot before, we can get 95% confidence interval:

- native (C++): (0.10579144538509686, 0.14679045461490317)
- WASM-based code: (0.5419682017893873, 1.228425798206798)

According to these execution time, we can see WASM-based code takes about 10 times longer to execute than the native code.

However, the data distribution does not conform to a normal distribution. The data distribution is shown below.

### Nativa Code Runtime Distribution:



#### WASM-based Code Runtime Distribution:

And we can also see that there are some unusually large values. In my analysis, the reason for such large differences could be that there are times when the memory is over-occupied by other processes during the running of the program, resulting in a few particularly irregular runtime values that are particularly large.

Since these problems may have caused errors in the mean and standard values, we can learn that the confidence intervals obtained can be considered statistically insignificant.

## Improve performance

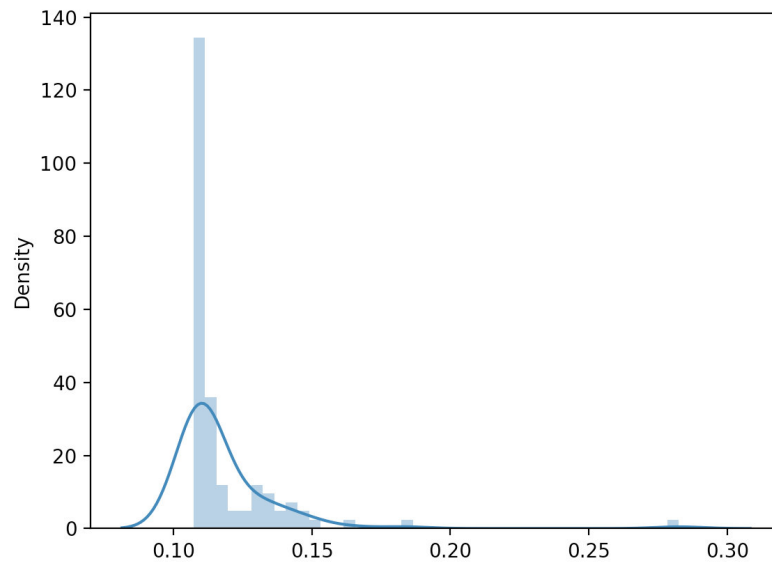
We can optimize by specifying the optimization flags, which are: `-O0`, `-O1`, `-O2`, `-Os`, `-Oz`, `-O3`. Here I choose `-O2` level optimization for compiled.

```
Fibonacci % emcc -O2 fib.cpp -o fib.js -s EXPORTED_FUNCTIONS=_getFib -s  
EXPORTED_RUNTIME_METHODS=ccall
```

After running this function 100 times and saving the output to `browser\_O2\_time.txt`, we can get its 95% confidence interval in the same way as before.

- native (C++): (0.10579144538509686, 0.14679045461490317)
- WASM-based code: (0.5419682017893873, 1.228425798206798)
- WASM-based code (-O2): (0.07599345381156607, 0.16062854619367917)

#### WASM-based Code (-O2) Runtime Distribution:



Although its distribution is also skewed and can be considered statistically insignificant, we can still learn that it is actually much faster than the WASM-based code without any optimization, and its performance is even close to that of the native code.