WebAssembly

WASM — In a Nutshell

about this guy

- Juhun “RangHo” Lee
- Sogang University Undergraduate
  - Dept. of Computer Science and Engineering
- Professional Procrastinator
- Commits bullshits for living

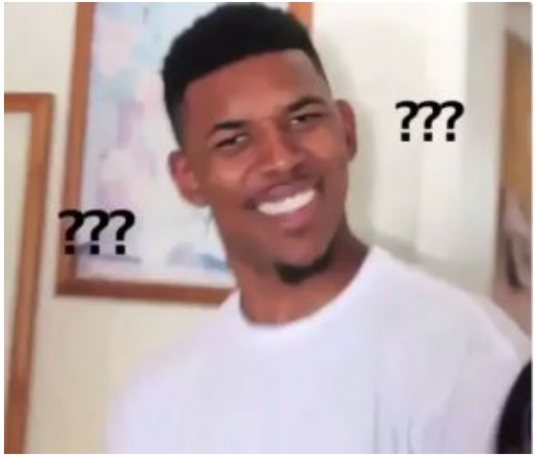
- Things I am interested in:
  - Programming Language Design
  - Computational Linguistics
  - System Programming
  - Game Programming
  - ...and many more

Find me at:

- Twitter **@RangHo_777**
- GitHub **@RangHo**

WebAssembly a.k.a. wasm

- WebAssembly = Web + Assembly
  - Web = Browser-based
  - Assembly = Low-level machine code
- Therefore, WebAssembly is a **browser-based low-level machine code.**



hold up right there
something’s not right

Machine code + Web browser = WTF

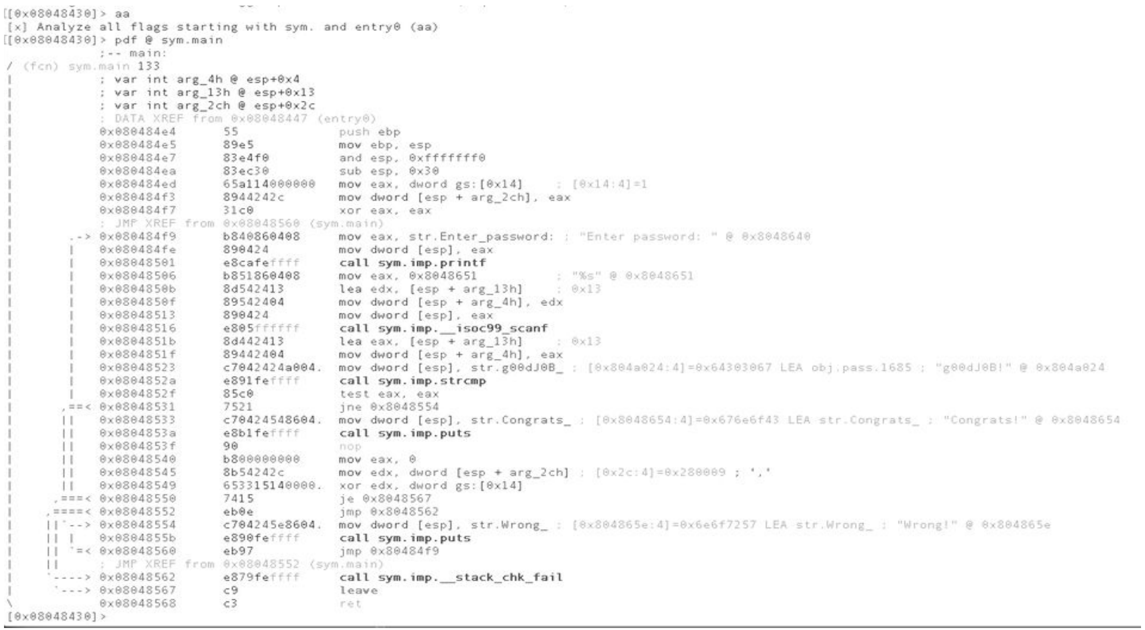
Assembly is notorious for having:

Unintelligible mess of instructions □

Code too raw to understand □

Weird data storage called *registers* □

Instruction pointer that jumps around like a 5-year-old child □



```
[[0x08048430]> aa
 [x] Analyze all flags starting with sym. and entry0 (aa)
[[0x08048430]> pdf @ sym.main
            ;-- main:
/ (fcn) sym.main 133
|           ; var int arg_4h @ esp+0x4
|           ; var int arg_13h @ esp+0x13
|           ; var int arg_2ch @ esp+0x2c
|           ; DATA XREF from 0x08048447 (entry0)
|           0x080484e4      55             push ebp
|           0x080484e5      89e5           mov ebp, esp
|           0x080484e7      83e4f0         and esp, 0xfffffff0
|           0x080484ea      83ec30         sub esp, 0x30
|           0x080484ed      65a114000000   mov eax, dword gs:[0x14]    ; [0x14:4]=1
|           0x080484f3      8944242c       mov dword [esp + arg_2ch], eax
|           0x080484f7      31c0           xor eax, eax
|           ; JMP XREF from 0x08048560 (sym.main)
|       .-> 0x080484f9      b840860408     mov eax, str.Enter_password: ; "Enter password: " @ 0x8048640
|       |   0x080484fe      890424         mov dword [esp], eax
|       |   0x08048501      e8cafeffff     call sym.imp.printf
|       |   0x08048506      b851860408     mov eax, 0x8048651          ; "%s" @ 0x8048651
|       |   0x0804850b      8d542413       lea edx, [esp + arg_13h]    ; 0x13
|       |   0x0804850f      89542404       mov dword [esp + arg_4h], edx
|       |   0x08048513      890424         mov dword [esp], eax
|       |   0x08048516      e805ffffff     call sym.imp.__isoc99_scanf
|       |   0x0804851b      8d442413       lea eax, [esp + arg_13h]    ; 0x13
|       |   0x0804851f      89442404       mov dword [esp + arg_4h], eax
|       |   0x08048523      c7042424a004.  mov dword [esp], str.g00dJ0B_ ; [0x804a024:4]=0x64303067 LEA obj.pass.1685 ; "g00dJ0B!" @ 0x804a024
|       |   0x0804852a      e891feffff     call sym.imp.strcmp
|       |   0x0804852f      85c0           test eax, eax
|      ,==< 0x08048531      7521           jne 0x8048554
|      ||   0x08048533      c70424548604.  mov dword [esp], str.Congrats_ ; [0x8048654:4]=0x676e6f43 LEA str.Congrats_ ; "Congrats!" @ 0x8048654
|      ||   0x0804853a      e8b1feffff     call sym.imp.puts
|      ||   0x0804853f      90             nop
|      ||   0x08048540      b800000000     mov eax, 0
|      ||   0x08048545      8b54242c       mov edx, dword [esp + arg_2ch] ; [0x2c:4]=0x280009 ; ','
|      ||   0x08048549      653315140000.  xor edx, dword gs:[0x14]
|     ,===< 0x08048550      7415           je 0x8048567
|    ,====< 0x08048552      eb0e           jmp 0x8048562
|    ||`--> 0x08048554      c704245e8604.  mov dword [esp], str.Wrong_ ; [0x804865e:4]=0x6e6f7257 LEA str.Wrong_ ; "Wrong!" @ 0x804865e
|    || |   0x0804855b      e890feffff     call sym.imp.puts
|    || `=< 0x08048560      eb97           jmp 0x80484f9
|    ||     ; JMP XREF from 0x08048552 (sym.main)
|    `----> 0x08048562      e879feffff     call sym.imp.__stack_chk_fail
|     `---> 0x08048567      c9             leave
\           0x08048568      c3             ret
[0x08048430]>
```

ok, what about wasm?

- WebAssembly is a **specification of a virtual machine** (VM)
  - JVM for Java, Kotlin, Scala, Groovy, etc.
  - CLR for C#, Visual Basic .NET, F#, etc.
  - And many many more (LLVM, Parrot, BEAM, Dalvik…)
- Maintained by W3C WebAssembly Working Group
- Compiled “assembly codes” are run in a sandbox of web browser
- Requires a “glue code” written in JavaScript
  - WebAssembly ⇔ JavaScript (DOM) ⇔ View

how wasm works

- WebAssembly is a **stack-based virtual machine**
  - Everything is put into a stack
  - cf. register-based virtual machine
  - e.g. JVM, CLR, Python VM, etc.
- 4 fundamental primitive types available
  - `i32` - 32-bit integer
  - `i64` - 64-bit integer
  - `f32` - 32-bit floating point decimal
  - `f64` - 64-bit floating point decimal
- Uses S-expressions to represent a code (like Lisp) (not really)

Pretty boring stuff...

web asm vs. *real* asm

```
(func $fibonacci (param $n i32) (result i32)

    (if (i32.eq (get_local $n) (i32.const 1))
        (then (return (i32.const 1))))
    (if (i32.eq (get_local $n) (i32.const 2))
        (then (return (i32.const 1))))

    (i32.add
        (call $fibonacci
            (i32.sub (get_local $n) (i32.const 1)))
        (call $fibonacci
            (i32.sub (get_local $n) (i32.const 2)))))
```

```
fibonacci:
    mov eax, [esp+4]
    cmp eax, 1
    ja fibonacci_recurse
    mov eax, 1
    ret

fibonacci_recurse:
    push ebx
    dec eax
    push eax
    call fibonacci
    mov ebx, eax
    dec [esp]
    add eax, ebx
    add esp, 4
    pop ebx
```

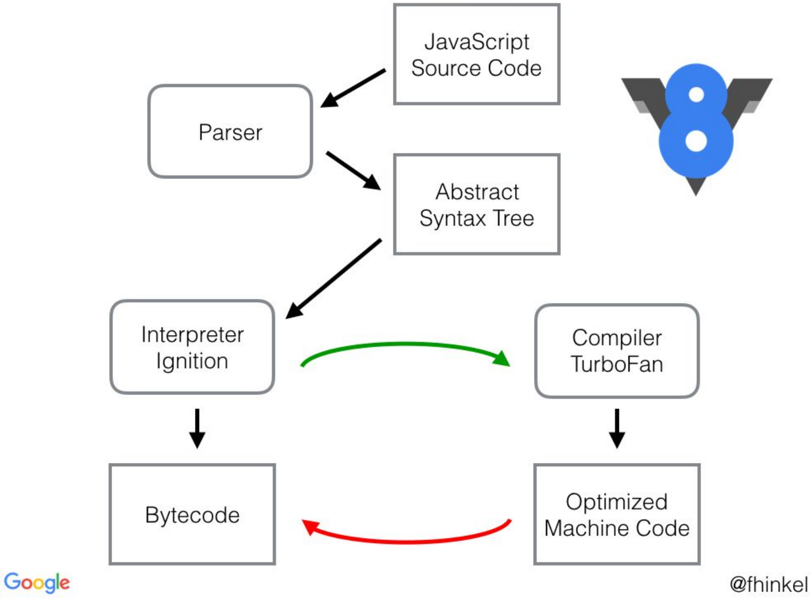
Past of WebAssembly

once upon a time...

- **JavaScript** is the only language that all major browsers support
  - JavaScript existed since Netscape
  - V8, SpiderMonkey, JavaScriptCore...
- Interpreted, dynamically typed language
  - Portability is awesome!
  - Yet there is a massive problem...

JavaScript is slow as hell.

if interpretation is too slow...



Ignition and **TurboFan** JIT Pipeline of Chrome's V8 Engine.

- "If interpretation is too slow, we can compile the code!"
- **Just-in-time Compilation**: Translation of frequently used code segment to machine code in order to improve performance

- JIT compiler makes many assumptions
- Here arises a new problem...

There are so many exceptions that JIT compiler becomes useless!

Humans cause inefficiencies.

Thus, *we don’t write* JavaScript.

two ways to solve this problem

The Firefox Way

The Chrome Way

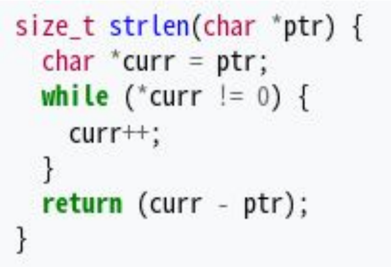
we don’t write JavaScript, we *target* it.

- Runtime type inference for JavaScript getting harder and harder

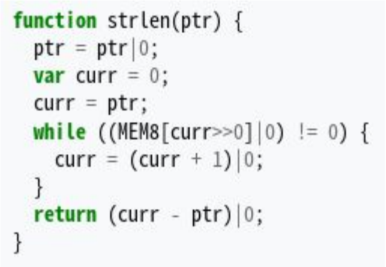
  “What if we *target* JavaScript from other statically typed languages…?”

  Project Emscripten and **asm.js**

- asm.js – a strict subset of JavaScript, where browsers can optimize before running
- Emscripten – LLVM backend that compiles C source code to asm.js



```
size_t strlen(char *ptr) {
  char *curr = ptr;
  while (*curr != 0) {
    curr++;
  }
  return (curr - ptr);
}
```



```
function strlen(ptr) {
  ptr = ptr|0;
  var curr = 0;
  curr = ptr;
  while ((MEM8[curr>>0]|0) != 0) {
    curr = (curr + 1)|0;
  }
  return (curr - ptr)|0;
}
```

we don’t write JavaScript, we go *native*.

Google Developers Live

November 14, 2013
Colt McAnlis hosts a conversation on
Chrome Native Client

- The fact that JavaScript is not compiled is an unfixable performance sacrifice

  “What if we run *native* code on web browsers…?”
  ~~aw shit here we go again~~

  Native Client a.k.a. NaCl

- NaCl – a set of C/C++ libraries that allows Chrome to run native binaries
- Failed spectacularly, switched to asm.js

not writing JS is good and all, but...

- We are at a point where compiling C code to JavaScript is seriously considered as a viable option
    - ~~System engineers: what the fuck~~

  “What if we define a virtual machine, and compile programming languages *for* that machine?”

  WebAssembly

- Basically a computer, but on a web browser
- Performance is on par with native binaries (!!!)

Present of WebAssembly

wasm is now stable!

- All major browsers now support WebAssembly out of the box
  - Reached cross-browser consensus on March 2017

Microsoft Edge	since Version 16 (October 2017)
Mozilla Firefox	since Version 52 (March 2017)
Google Chrome	since Version 57 (March 2017)
Apple Safari	since Version 11 (September 2017)
Opera Browser	since Version 44 (March 2017)
~~Internet Explorer~~	~~Not supported. What did you expect?~~

wasm, the brand-new and hipster version of Java

- Not only browsers, Node.js started supporting WebAssembly
  - Current method of importing WebAssembly
    - Read .wasm file
    - Instantiate WebAssembly VM
    - Create and populate shared memory
  - Experimental method of importing WebAssembly
    - `import <component> as <name> from "/path/to/wasm";`
- Cross-platform, JavaScript-based programming is possible

name your favourite; it’s probably supported

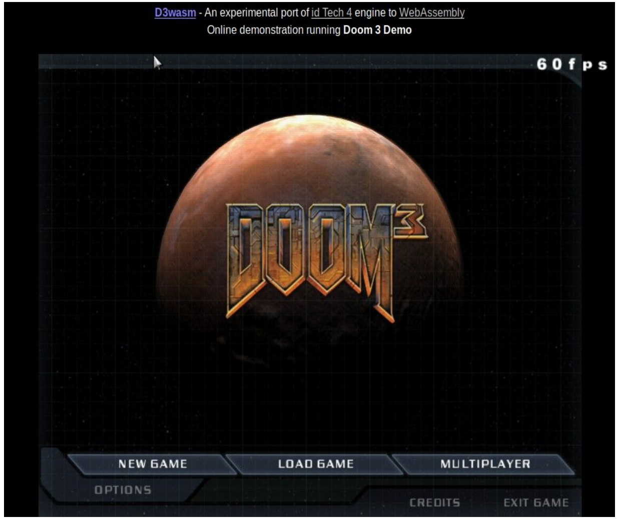
Some of languages that can be used with (or compiled to) WebAssembly

● C/C++	with Emscripten
● Kotlin	with Kotlin/Native
● Swift	with SwiftWasm
● C#	with Mono or Uno Platform or Blazor
● Java	TeaVM
● Python	Pyodide
● PHP	PIB
● Rust	with the official compiler (`rustc`)
	□ More about this later

say goodbye to flash games

- WebAssembly is already performant enough to run games
  - Although internet speed plays a huge role in game playability
  - Poor internet connection = it takes eternity to load a game
- In case of Unity:
  - Started supporting browser-based games via external program
  - Migrated to JavaScript-based WebGL player
  - Currently in progress of migrating to WebAssembly player
- In case of Unreal Engine
  - UE4 started supporting HTML5 build since March 2017
  - Showcased Zen Garden demo, originally developed for Metal API

let’s try out some demos



D3wasm - An experimental port of id Tech 4 engine to WebAssembly

Online demonstration running **Doom 3 Demo**

WebAssembly port of Doom, friend of all programmers in the world.

Funky Karts, a kart game build ground-up from C++, targeting WebAssembly.

When ask about wasm, they always talk about Rust.

What about it?

what is rust?

- New programming language!
  - First version appeared on July of 2010
- Actively developed by Mozilla Foundation
- StackOverflow’s “most loved programming language” winner since 2016
- Object-oriented + Functional paradigm
- Designed to replace C/C++

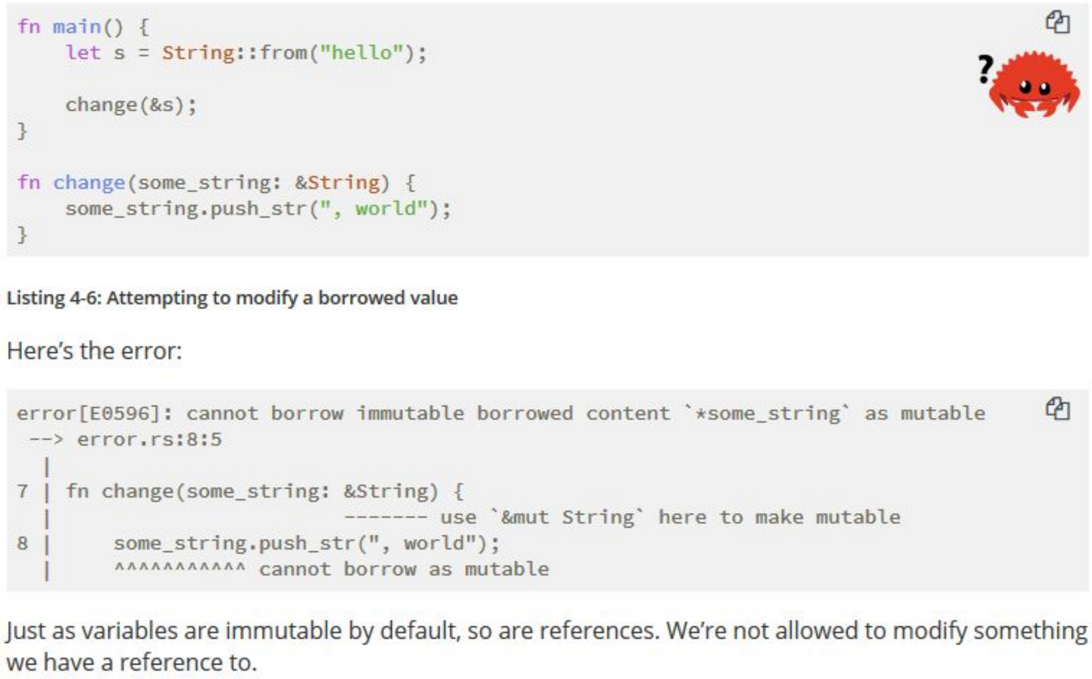
- **Guarantees memory safety at compile time** (= no segmentation fault)

guarantee of raw memory safety???

- Explicit “ownership” of values
- Extensive “Borrow Checker™” that manages ownership

  “I will make sure that no one touches your values!”

- This eliminates most memory errors caused by ownership mismatch
- No need to `malloc()` and `free()` memories!



```
fn main() {
    let s = String::from("hello");

    change(&s);
}

fn change(some_string: &String) {
    some_string.push_str(", world");
}
```

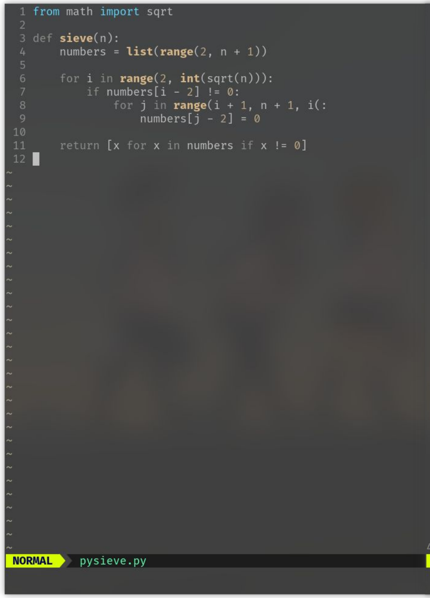
Listing 4-6: Attempting to modify a borrowed value

Here’s the error:

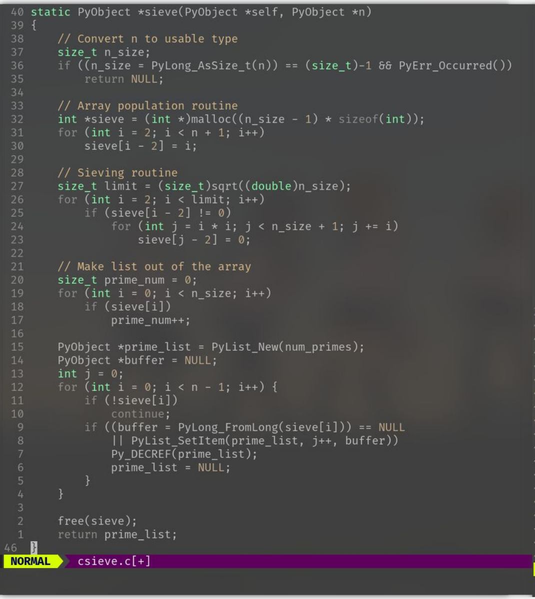
```
error[E0596]: cannot borrow immutable borrowed content `*some_string` as mutable
 --> error.rs:8:5
  |
7 | fn change(some_string: &String) {
  |                        ------- use `&mut String` here to make mutable
8 |     some_string.push_str(", world");
  |     ^^^^^^^^^^^ cannot borrow as mutable
```

Just as variables are immutable by default, so are references. We’re not allowed to modify something we have a reference to.

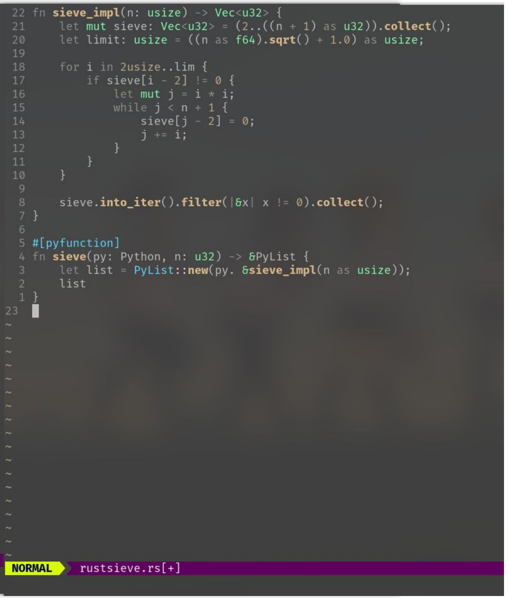
what about performance?



```python
from math import sqrt

def sieve(n):
    numbers = list(range(2, n + 1))

    for i in range(2, int(sqrt(n))):
        if numbers[i - 2] != 0:
            for j in range(i + 1, n + 1, i(:
                numbers[j - 2] = 0

    return [x for x in numbers if x != 0]
```



```c
static PyObject *sieve(PyObject *self, PyObject *n)
{
    // Convert n to usable type
    size_t n_size;
    if ((n_size = PyLong_AsSize_t(n)) == (size_t)-1 && PyErr_Occurred())
        return NULL;

    // Array population routine
    int *sieve = (int *)malloc((n_size - 1) * sizeof(int));
    for (int i = 2; i < n + 1; i++)
        sieve[i - 2] = i;

    // Sieving routine
    size_t limit = (size_t)sqrt((double)n_size);
    for (int i = 2; i < limit; i++)
        if (sieve[i - 2] != 0)
            for (int j = i * i; j < n_size + 1; j += i)
                sieve[j - 2] = 0;

    // Make list out of the array
    size_t prime_num = 0;
    for (int i = 0; i < n_size; i++)
        if (sieve[i])
            prime_num++;

    PyObject *prime_list = PyList_New(num_primes);
    PyObject *buffer = NULL;
    int j = 0;
    for (int i = 0; i < n - 1; i++) {
        if (!sieve[i])
            continue;
        if ((buffer = PyLong_FromLong(sieve[i])) == NULL
            || PyList_SetItem(prime_list, j++, buffer))
            Py_DECREF(prime_list);
            prime_list = NULL;
        }
    }

    free(sieve);
    return prime_list;
}
```



```rust
fn sieve_impl(n: usize) -> Vec<u32> {
    let mut sieve: Vec<u32> = (2..((n + 1) as u32)).collect();
    let limit: usize = ((n as f64).sqrt() + 1.0) as usize;

    for i in 2usize..lim {
        if sieve[i - 2] != 0 {
            let mut j = i * i;
            while j < n + 1 {
                sieve[j - 2] = 0;
                j += i;
            }
        }
    }

    sieve.into_iter().filter(|&x| x != 0).collect();
}

#[pyfunction]
fn sieve(py: Python, n: u32) -> &PyList {
    let list = PyList::new(py. &sieve_impl(n as usize));
    list
}
```

□ Python 3

25ms to process

100,000 numbers

□ C/C++

700µs to process

100,000 numbers

□ Rust

670µs to process

100,000 numbers

why rust is a big deal for wasm

- Rust is one of the first languages to support WebAssembly
- Existing Rust programs can be easily compiled to WebAssembly
    - `wasm_bindgen` crate generates JavaScript bind source code
- LLVM-based compiler toolchain

- Currently, best languages to create WebAssembly binaries are:
    - C/C++
    - AssemblyScript (subset of TypeScript)
    - **Rust**

why don’t we try out right now?

https://webassembly.studio/

Future of WebAssembly

wasm is not complete by any means

- Although WebAssembly is stable, it requires many improvements
- Current limitations:
  - Standards are fragmented into two different branches
  - Only types with fixed length can be sent as function parameters
  - Nonexistent threading
  - Exception Handler
  - Reference type is missing
  - Garbage Collection might be needed for higher-level operations
  - SIMD causes unnecessary overhead
  - **WebAssembly DOM API**

still it is usable, right?

- WebAssembly Binary Version is frozen at `0x01`
  - Current specification is final
  - Other features are added in a backwards-compatible manner
- Fast enough to run 3D games that draw scenes in `<canvas>`
- More and more languages start targeting WebAssembly
  - https://github.com/appcypher/awesome-wasm-langs
- Web programming is slowly diverting from JavaScript
  - Microsoft's Blazor makes C# as the main scripting engine
  - JavaScript virtual DOM + C++/gccx = WebAssembly single-page web app (?!)

if I dare make speculations…

- WebAssembly will be the *new Java*
  - Cross-platform
  - Extendable
  - ~~I mean Java already runs on WebAssembly…~~
- Browser-based gaming is possible
  - Streaming is yet inaccessible to most people (e.g. Google Stadia)
  - With new APIs like WebGPU coming soon, WebAssembly will create new markets

Projects I am working on

project
make pini great again

- PiniEngine ~~is~~ was a visual novel engine based on cocos2d-x
    - The company developing this went bankrupt
    - “We leave PiniEngine to the Open Source community” ~~what~~
- Korean-based scripting DSL
    - Pretty revolutionary
- Codebase is fascinating

- Complete C++ reimplementation
- Web player with WebAssembly

https://github.com/RangHo/pini-engine

project scratch2wasm

- Scratch is a block-based educational programming language
- Powerful enough to implement a parser

  “What if I make a Scratch compiler that targets real ELF binary…?”

- It’s more like LLVM’s scratch frontend
- LLVM-based? WebAssembly!

- It’s a joke project but hey it’s funny

https://github.com/RangHo/scratchc

Questions & Answers

Thank you.

Additional questions?

hello@rangho.me