

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

```
[0x0040430e]> aa
[*] Analyze all flags starting with sym. and entry0 (aa)
[[0x0040430e]> 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 0x00404447 (entry0)
|   0x004044e4 55          push ebp
|   0x004044e5 89e5        mov ebp, esp
|   0x004044e7 83e4f0      and esp, 0xfffff0
|   0x004044ea 83ec30      sub esp, 0x30
|   0x004044ed 65a1140000 mov eax, dword gs:[0x14] ; [0x14:4]=1
|   0x004044f3 8944242c    mov dword [esp + arg_2ch], eax
|   0x004044f7 31c0        xor eax, eax
|   ; JMP XREF from 0x00404560 (sym.main)
|   --> 0x004044f9 b44860408   mov eax, str.Enter_password; "Enter password: " @ 0x00408640
|   | 0x004044fe 896424      mov dword [esp], eax
|   | 0x00404501 e8c4e7ffff  call sym.imp.printf
|   | 0x00404506 b851860408  mov eax, 0x00408651 ; "%s" @ 0x00408651
|   | 0x0040450b 8d542413    lea edx, [esp + arg_13h] ; 0x13
|   | 0x0040450f 89542404    mov dword [esp + arg_4h], edx
|   | 0x00404513 896424      mov dword [esp], eax
|   | 0x00404516 e805ffff    call sym.imp._isoc99_scanf
|   | 0x0040451b 8d442413    lea eax, [esp + arg_13h] ; 0x13
|   | 0x0040451f 89442404    mov dword [esp + arg_4h], eax
|   | 0x00404523 c7042424a004. mov dword [esp], str.g00dJ0B_ ; [0x0040a024:4]=0x64303067 LEA obj.pass.1685: "g00dJ0B!" @ 0x0040a024
|   | 0x0040452a e891feffff  call sym.imp.strcmp
|   | 0x0040452f 85c0        test eax, eax
|   | 0x00404531 jne 0x00408554
|   | 0x00404533 c70424548604. mov dword [esp], str.Congrats_ ; [0x00408654:4]=0x67606f43 LEA str.Congrats_: "Congrats!" @ 0x00408654
|   | 0x0040453a e8b1feffff  call sym.imp.puts
|   | 0x0040453f 90          nop
|   | 0x00404540 b800000000  mov eax, 0
|   | 0x00404545 8b54242c    mov edx, dword [esp + arg_2ch]; [0x2c:4]=0x200009 ; ', '
|   | 0x00404549 653315140000. xor edx, dword gs:[0x14]
|   | 0x00404550 7415        je 0x00408567
|   | 0x00404552 eb0e        jmp 0x00408562
|   | 0x00404554 c704245e8604. mov dword [esp], str.Wrong_ ; [0x0040865e:4]=0x6e6f7257 LEA str.Wrong_: "Wrong!" @ 0x0040865e
|   | 0x0040455b e899feffff  call sym.imp.puts
|   | 0x00404560 eb97        jmp 0x004044f9
|   | ; JMP XREF from 0x00404552 (sym.main)
|   | 0x00404562 e879feffff  call sym.imp.__stack_chk_fail
|   | 0x00404567 c9          leave
|   | 0x00404568 c3          ret
[0x0040430e]>
```

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 \Leftrightarrow JavaScript (DOM) \Leftrightarrow 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
```

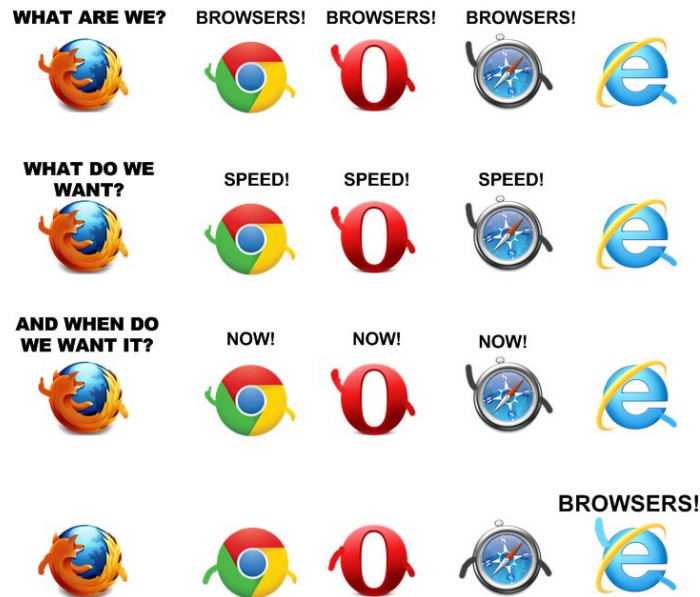
Past of WebAssembly

once upon a time...

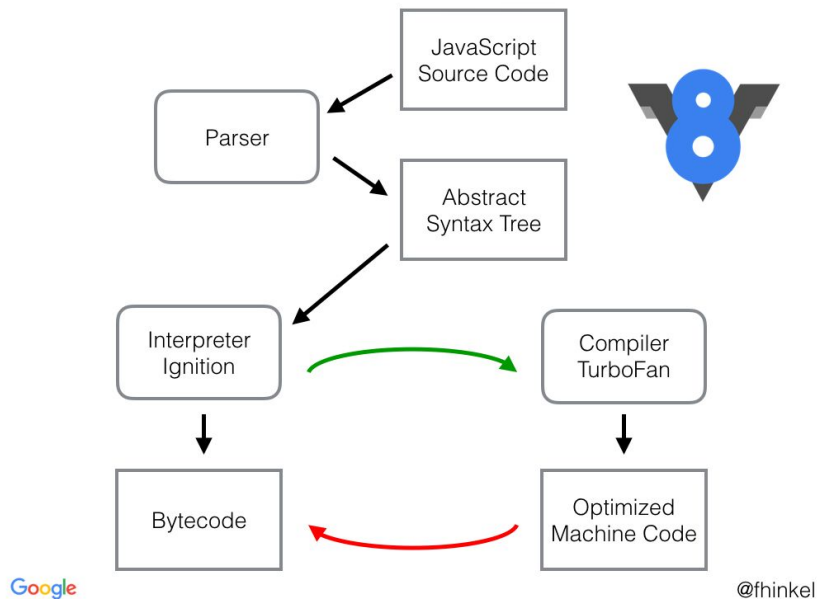
- **JavaScript** is the only language that all major browsers support
 - JavaScript existed since Netscape
 - V8, SpiderMonkey, JavaScriptCore...
- Interpreted, dynamically typed language
 - Portability is awesome!
 - Yet there is a massive problem...

JavaScript is slow as hell.

The Usual Suspects



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

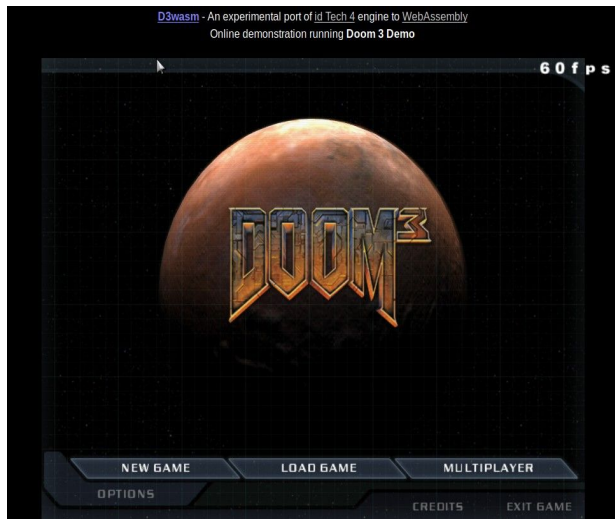
Some of languages that can be used with (or compiled to) WebAssembly

- 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



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?

```
1 from math import sqrt
2
3 def sieve(n):
4     numbers = list(range(2, n + 1))
5     for i in range(2, int(sqrt(n))):
6         if numbers[i - 2] != 0:
7             for j in range(i + 1, n + 1, i):
8                 numbers[j - 2] = 0
9     return [x for x in numbers if x != 0]
```

```
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

NORMAL pysieve.py

```
40 static PyObject *sieve(PyObject *self, PyObject *n)
41 {
42     // Convert n to usable type
43     size_t n_size;
44     if ((n_size = PyLong_AsSize_t(n)) == (size_t)-1 && PyErr_Occurred())
45         return NULL;
46
47     // Array population routine
48     int *sieve = (int *)malloc((n_size - 1) * sizeof(int));
49     for (int i = 2; i < n + 1; i++)
50         sieve[i - 2] = 1;
51
52     // Sieving routine
53     size_t limit = (size_t)sqrt((double)n_size);
54     for (int i = 2; i < limit; i++)
55         if (sieve[i - 2] != 0)
56             for (int j = i * i; j < n_size + 1; j += i)
57                 sieve[j - 2] = 0;
58
59     // Make list out of the array
60     size_t prime_num = 0;
61     for (int i = 0; i < n_size; i++)
62         if (sieve[i])
63             prime_num++;
64
65     PyObject *prime_list = PyList_New(num_primes);
66     PyObject *buffer = NULL;
67     int j = 0;
68     for (int i = 0; i < n - 1; i++) {
69         if (sieve[i])
70             continue;
71         if ((buffer = PyLong_FromLong(sieve[i])) == NULL
72             || PyList_SetItem(prime_list, j++, buffer)
73             || Py_DECREF(buffer))
74             prime_list = NULL;
75     }
76
77     free(sieve);
78     return prime_list;
79 }
```

NORMAL csieve.c[+]

```
22 fn sieve_impl(n: usize) -> Vec<u32> {
23     let mut sieve: Vec<u32> = (2..((n + 1) as u32)).collect();
24     let limit: usize = ((n as f64).sqrt() + 1.0) as usize;
25
26     for i in 2..size::lim {
27         if sieve[i - 2] != 0 {
28             let mut j = i * i;
29             while j < n + 1 {
30                 sieve[j - 2] = 0;
31                 j += i;
32             }
33         }
34     }
35
36     sieve.into_iter().filter(|&x| x != 0).collect();
37 }
38
39 #[pyfunction]
40 fn sieve(py: Python, n: u32) -> &PyList {
41     let list = PyList::new(py, &sieve_impl(n as usize));
42     list
43 }
```

NORMAL rustsieve.rs[+]

□ 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