

# What's Old is New Again

## An Introduction to WebAssembly



# Guy Royse

Developer Evangelist  
DataRobot

 @guyroyse

 [github.com/guyroyse](https://github.com/guyroyse)



# What is WebAssembly?

(Hint. It's a solution to a problem.)



# WTH, JavaScript?

```
[] = ![]; // → true
```

```
NaN === NaN; // → false
```

```
Number.MIN_VALUE > 0; // → true
```

```
parseInt("firetruck"); // → NaN  
parseInt("firetruck", 16); // → 15
```

```
console.log.call.call.call.call  
    .call.apply(a ⇒ a, [1, 2]);
```

```
Math.min() > Math.max(); // → true
```

# JS



The image shows a vast collection of identical yellow rubber ducks. They are arranged in neat, diagonal rows on multiple shelves, creating a strong sense of repetition and abundance. The ducks are bright yellow with red beaks and small blue eyes. The perspective is from a low angle, looking down the length of the shelves, which makes the rows appear to converge in the distance. The overall lighting is warm and slightly soft, emphasizing the uniformity and sheer quantity of the objects.

**So Many Choices**

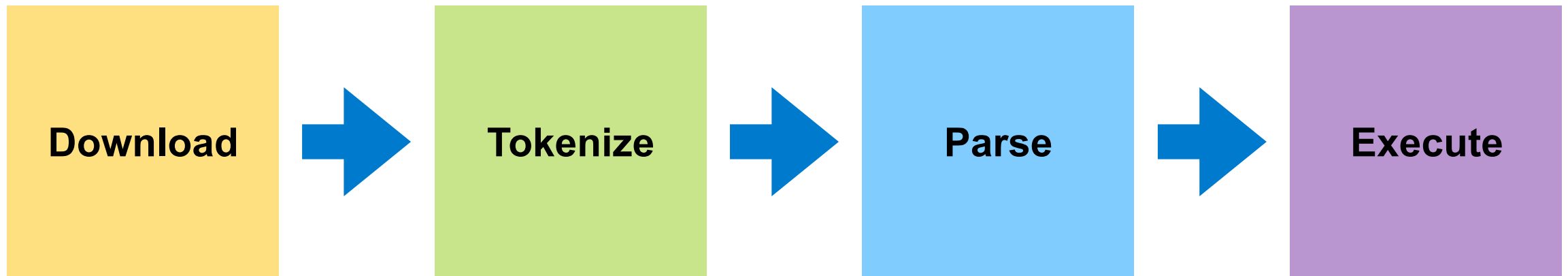





# Modern JavaScript Environment

It's easy! Just use these simple tools:

- npm or yarn
- Bower
- Webpack
- Babel
- Grunt or Gulp
- live-server
- React and Redux
- Mocha, Chai, and Sinon





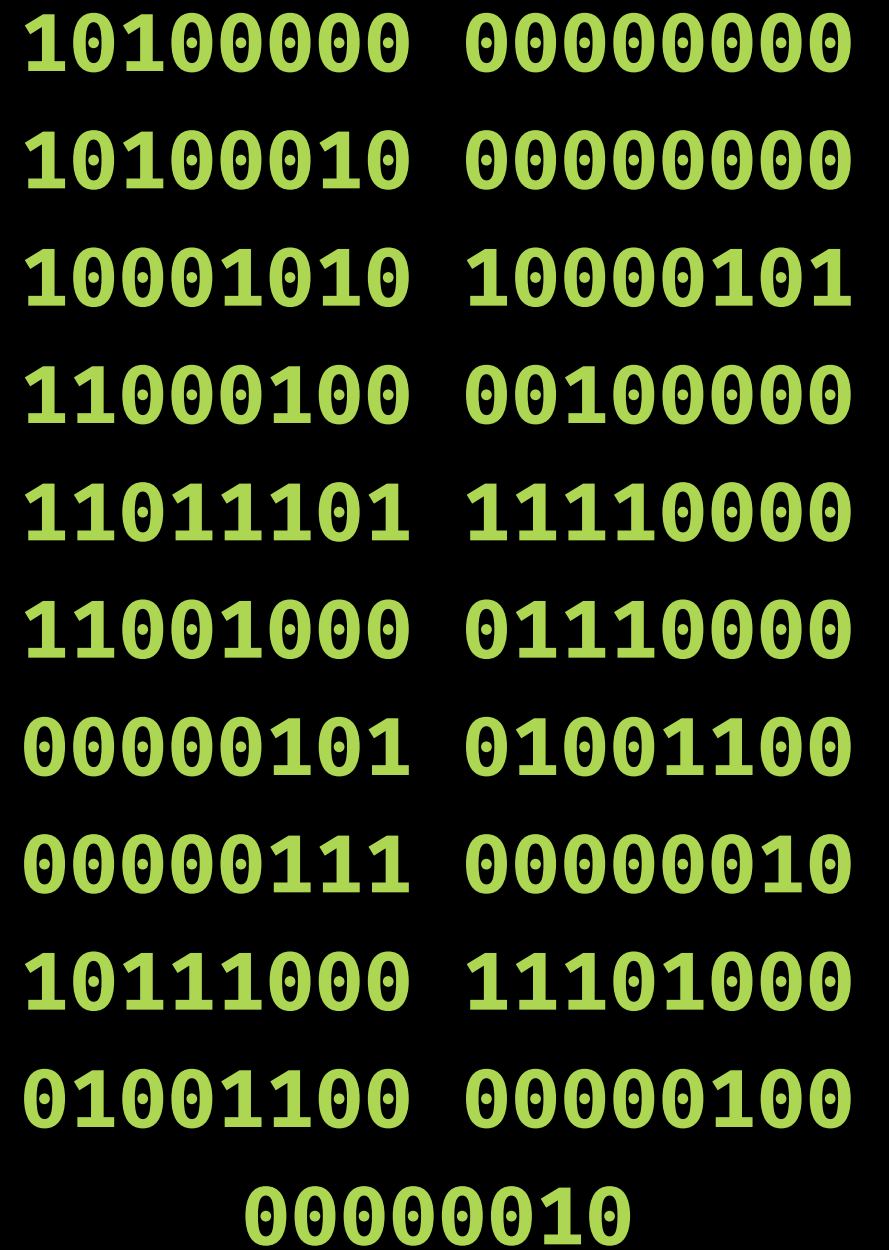
The background of the image is a blurred, dark field filled with lines of computer code in various colors (white, yellow, green, blue, red). The code appears to be a mix of different programming languages, including JavaScript and CSS. The text is centered and reads:

**What If You  
Could Compile  
Code for the  
Browser?**





**WA**







Address	Binary	Hex	Assembly
0200	10100000 00000000	A0 00	LDY #\$00
0202	10100010 00000000	A2 00	LDX #\$00
0204	10001010	8A	LOOP2 TXA
0205	10000101 11000100	85 C4	STA \$00C4
0207	00100000 11011101 11110000	20 22 F0	LOOP1 JSR SCAN
020A	11001000	C8	INY
020B	01110000 00000101	70 05	BVS RESET
020D	01001100 00000111 00000010	4C 07 02	JMP LOOP1
0210	10111000	B8	RESET CLV
0211	11101000	E8	INX
0212	01001100 00000100 00000010	4C 04 02	JMP LOOP2



*In the 1950's von Neumann was employed as a consultant to review proposed and ongoing advanced technology projects. One day a week, von Neumann "held court" at 590 Madison Avenue, New York. On one of these occasions in 1954 he was confronted with the FORTRAN concept; John Backus remembered von Neumann being unimpressed and that he asked "why would you want more than machine language?" Frank Beckman, who was also present, recalled that von Neumann dismissed the whole development as "but an application of the idea of Turing's 'short code'." Donald Gilles, one of von Neumann's students at Princeton, and later a faculty member at the University of Illinois, recalled in the mid-1970's that the graduates students were being "used" to hand assemble programs into binary for their early machine (probably the IAS machine). He took time out to build an assembler, but when von Neumann found out about he was very angry, saying (paraphrased), "It is a waste of a valuable scientific computing instrument to use it to do clerical work." (source: [John von Neuman and von Neumann Architecture for Computers \(1945\)](#))*





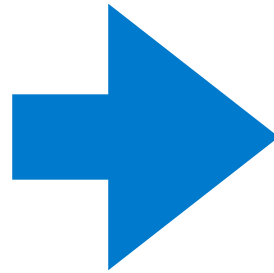
LDY #100  
 LDX #100  
 LOOP2 TXA  
 STA \$00C4  
 LOOP1 JSR SCAN  
 INY  
 BVS RESET  
 JMP LOOP1  
 RESET CLV  
 INX  
 JMP LOOP2

0200	A0	<del>00</del>
0202	A2	<del>00</del>
0204	8A	
0205	85	C4
0207	70	22 F0
020A	C8	
020B	70	05
020D	4C	07 02
0210	B8	
0211	E8	
0212	4C	04 02



```
int x = 0;
int y = 0;

for (x=0;x<12;x++)
{
    for (y=0;y<12;y++)
    {
        scan(x, y);
    }
}
```



```
10100000 00000000
10100010 00000000
10001010 10000101
11000100 00100000
11011101 11110000
11001000 01110000
00000101 01001100
00000111 00000010
10111000 11101000
01001100 00000100
00000010
```

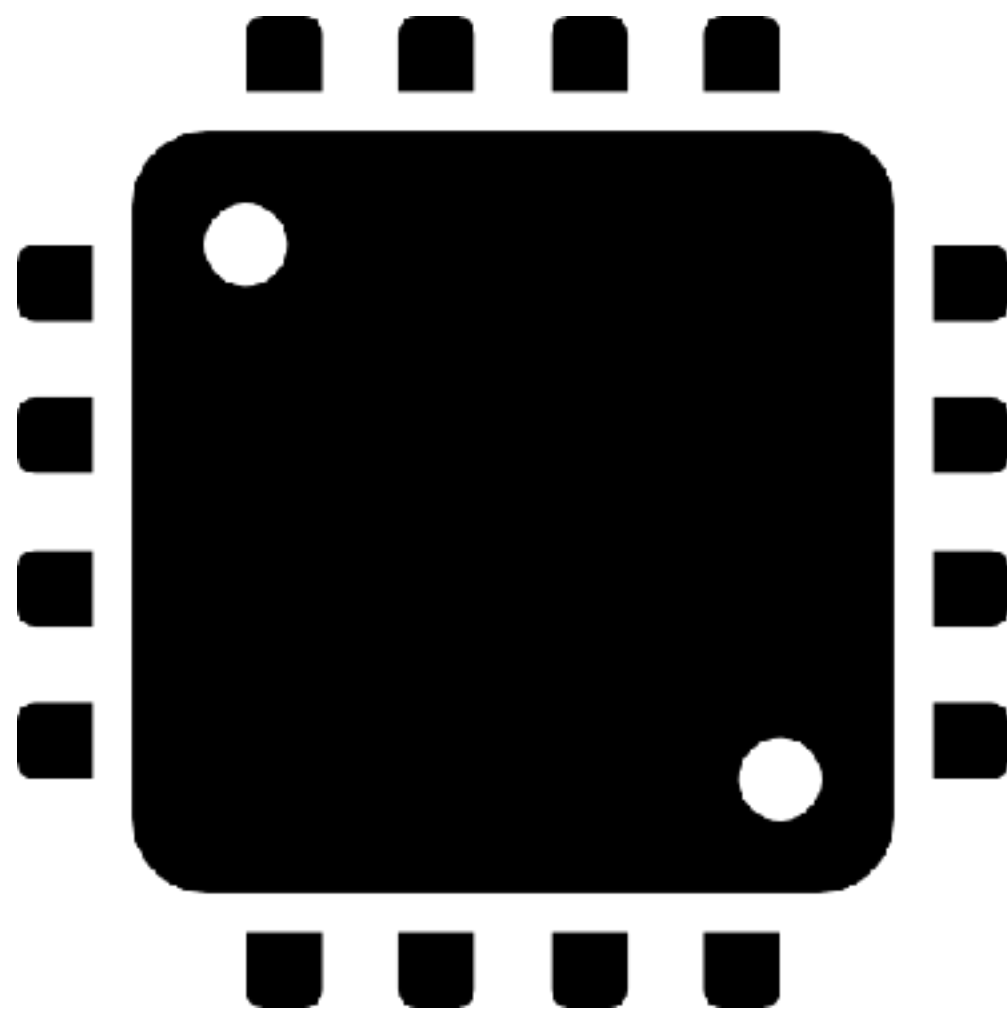










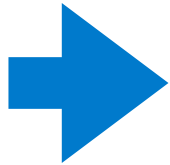


# How Virtual Machines Work

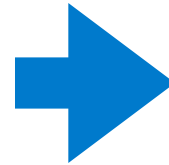
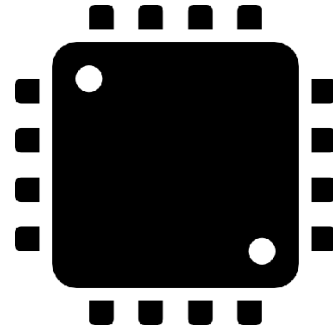
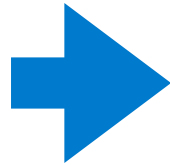


```
int x = 0;
int y = 0;

for (x=0;x<12;x++)
{
    for (y=0;y<12;y++)
    {
        scan(x, y);
    }
}
```



```
CA FE BA BE 00 00 00 31 00
13 07 00 10 07 00 11 01 00
07 63 6F 6E 76 65 72 74 01
00 3A 28 4C 6A 61 76 61 2F
6C 61 6E 67 2F 49 74 65 72
61 62 6C 65 3B 4C 6A 61 76
61 2F 6C 61 6E 67 2F 4F 62
6A 65 63 74 3B 29 4C 6A 61
76 61 2F 6C 61 6E 67 2F 4F
62 6A 65 63 74 3B 01 00 0A
45 78 63 65 70 74 69 6F 6E
73 07 00 12 01 00 09 53 69
```



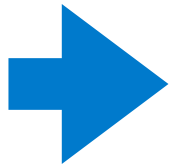


# How WebAssembly Works

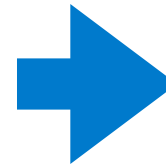
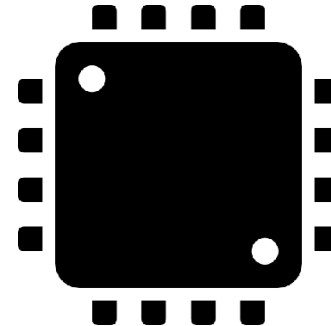
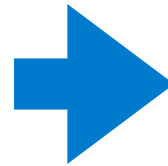


```
int x = 0;
int y = 0;

for (x=0;x<12;x++)
{
  for (y=0;y<12;y++)
  {
    scan(x, y);
  }
}
```



```
00 61 73 6D 01 00 00 00 01
10 03 60 01 7F 00 60 01 7F
01 7F 60 02 7F 7F 01 7F 02
15 01 08 66 69 7A 7A 62 75
7A 7A 08 63 61 6C 6C 62 61
63 6B 00 00 03 03 02 01 02
07 0C 01 08 66 69 7A 7A 62
75 7A 7A 00 01 0A 44 02 39
00 20 00 41 0F 10 02 04 40
41 7D 10 00 41 7D 0F 0B 20
00 41 05 10 02 04 40 41 7E
10 00 41 7E 0F 0B 20 00 41
```



# WebAssembly in the Browser



## Browser

### CSS

```
.effort {  
  display: none  
}
```

### HTML

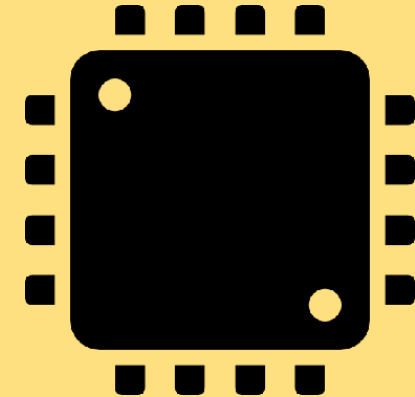
```
<html>  
  ...  
</html>
```

### WASM

```
00 61 73 6D 01 00 00  
00 01 10 03 60 01 7F  
00 60 01 7F 01 7F 60  
02 7F 7F 01 7F 02 15  
01 08 66 69 7A 7A 62  
75 7A 7A 08 63 61 6C  
6C 62 61 63 6B 00 00  
03 03 02 01 02 07 0C  
01 08 66 69 7A 7A 62  
75 7A 7A 00 01 0A 44  
02 39 00 20 00 41 0F  
10 02 04 40 41 7D 10  
00 41 7D 0F 0B 20 00  
41 05 10 02 04 40 41  
7E 10 00 41 7E 0F 0B  
20 00 41 03 10 02 04  
40 41 7F 10 00 41 7F  
0F 0B 20 00 10 00 20
```

### JavaScript

```
return fetch('foo.wasm')  
  .then(r => {  
    return r.arrayBuffer()  
  })  
  .then(bytes => {  
    return WebAssembly.instantiate(bytes)  
  })
```





# WebAssembly Text Format



```
00 61 73 6D 01 00 00
00 01 10 03 60 01 7F
00 60 01 7F 01 7F 60
02 7F 7F 01 7F 02 15
01 08 66 69 7A 7A 62
75 7A 7A 08 63 61 6C
6C 62 61 63 6B 00 00
03 03 02 01 02 07 0C
01 08 66 69 7A 7A 62
75 7A 7A 00 01 0A 44
02 39 00 20 00 41 0F
10 02 04 40 41 7D 10
00 41 7D 0F 0B 20 00
41 05 10 02 04 40 41
7E 10 00 41 7E 0F 0B
20 00 41 03 10 02 04
40 41 7F 10 00 41 7F
0F 0B 20 00 10 00 20
```

```
(module

  (export "add" (func $add))

  (func $add (param $a i32) (param $b i32) (result i32)
    get_local $a
    get_local $b
    i32.add
  )
)
```

# WebAssembly is Stack-based



```
(module  
  (export "add" (func $add))  
  
  (func $add (param $a i32) (param $b i32) (result i32)  
    get_local $a  
    get_local $b  
    i32.add  
  )  
)
```

Stack



# Push an Argument

```
(module  
  
  (export "add" (func $add))  
  
  (func $add (param $a i32) (param $b i32) (result i32)  
    get_local $a  
    get_local $b  
    i32.add  
  )  
)
```

5

Stack





# Push Another Argument

```
(module

  (export "add" (func $add))

  (func $add (param $a i32) (param $b i32) (result i32)
    get_local $a
    get_local $b
    i32.add
  )
)
```

10

5

Stack



# Pop the Arguments

```
(module  
  
  (export "add" (func $add))  
  
  (func $add (param $a i32) (param $b i32) (result i32)  
    get_local $a  
    get_local $b  
    i32.add  
  )  
)
```

10

5

Stack

# Add & Push



15

Stack

```
(module

  (export "add" (func $add))

  (func $add (param $a i32) (param $b i32) (result i32)
    get_local $a ;; 5
    get_local $b ;; 10
    i32.add
  )
)
```



# Instantiating Modules



```
(module  
  (export "add" (func $add))  
  
  (func $add  
    (param $a i32)  
    (param $b i32)  
    (result i32)  
    get_local $a  
    get_local $b  
    i32.add  
  )  
)
```

```
fetch('sample.wasm')  
  .then(response => {  
    return response.arrayBuffer()  
  })  
  .then(bytes => {  
    return WebAssembly.instantiate(bytes)  
  })  
  .then(module => {  
    // use webassembly module  
  })
```

# Instantiating Modules the Better Way



```
(module  
  (export "add" (func $add))  
  
  (func $add  
    (param $a i32)  
    (param $b i32)  
    (result i32)  
    get_local $a  
    get_local $b  
    i32.add  
  )  
)
```

```
WebAssembly.instantiateStreaming(  
  fetch('sample.wasm')).then(module => {  
    // use webassembly module  
  })
```

# Exporting Functions



```
(module  
  (export "add" (func $add))  
  
  (func $add  
    (param $a i32)  
    (param $b i32)  
    (result i32)  
    get_local $a  
    get_local $b  
    i32.add  
  )  
)
```

```
WebAssembly.instantiateStreaming(  
  fetch('sample.wasm')).then(module => {  
    let x = module.instance.exports.add(5, 10)  
    let y = module.instance.exports.add(x, 10)  
    console.log(x, y)  
  })
```



# Importing Functions



```
(module

  (import "js" "logCallback"
    (func $logCallback)
    (param i32)
  )

  (export "log" (func $log))

  (func $log (param $a i32)
    get_local $a
    call $logCallback
  )
)
```

```
let imports = {
  js: {
    logCallback: x => console.log(x)
  }
}

WebAssembly.instantiateStreaming(
  fetch('sample.wasm'), imports).then(module => {
  module.instance.exports.log(5)
  module.instance.exports.log(10)
})
```

# Start Function



```
(module

  (import "js" "logCallback"
    (func $logCallback)
    (param i32)
  )

  (start $main)

  (func $main
    i32.const 42
    call $logCallback
  )
)
```

```
let imports = {
  js: {
    logCallback: x => console.log(x)
  }
}

WebAssembly.instantiateStreaming(
  fetch('sample.wasm'), imports).then(module => {
  console.log('It started!')
})
```

# Shared Memory



```
(module

  (import "js" "memory"
    (memory 1)
  )

  (data (i32.const 0) 13)

  (start $main)

  (func $main
    i32.const 1    ;; position 1
    i32.const 23   ;; value of 23
    i32.store8     ;; store it
  )
)
```

```
let memory = new WebAssembly.Memory({ initial: 1 })

let imports = { js: { memory: memory } }

WebAssembly.instantiateStreaming(
  fetch('sample.wasm'), imports).then(module => {
    let shared = new Uint8Array(memory.buffer)

    console.log(shared[0]) // fetches 13
    console.log(shared[1]) // fetches 23

    shared[3] = 42          // stores 42 at position 3
  })
```



# Storing Strings



```
(module

  (import "js" "memory"
    (memory 1)
  )

  (data (i32.const 0) "\ud83d\udc1f\000")

  (export "hello" (func $hello))

  (func $hello (result i32)
    i32.const 0
  )
)
```

```
let decoder = new TextDecoder('utf8')

let memory = new WebAssembly.Memory({ initial: 1 })
let imports = { js: { memory: memory } }

WebAssembly.instantiateStreaming(
  fetch('sample.wasm'), imports).then(module => {
    let position = module.instance.exports.hello()
    let shared = new Uint8Array(memory.buffer, position)
    let index = shared.indexOf(0)

    let bytes = shared
    if (index !== -1) bytes = shared.slice(0, index)

    let string = decoder.decode(bytes)
    console.log(string)
  })
```

# Other Stuff



## Locals

```
(func $foo
  (result i32)

  (local $n i32)

  i32.const 42
  set_local $n

  get_local $n

  tee_local $n
)
```

## Globals

```
(global $n
  (import
    "js"
    "global")
  (mut i32))

(func $foo
  (result i32)

  i32.const 42
  set_global $n

  get_global $n
)
```

## S-Expressions

```
(func $foo
  (result i32)

  (i32.add
    (i32.const 5)
    (i32.const 10)
  )

)
```

## Tables

```
(table 3 anyfunc)

(elem (i32.const 0)
  $foo)

(elem (i32.const 1)
  $bar $baz)

(func $foo)
(func $bar)
(func $baz)
```

# Other Languages



## **Awesome Wasm Languages**

<https://github.com/appcypher/awesome-wasm-langs>



# Demo



# Resources



- [WebAssembly.org](https://webassembly.org) - official site for WebAssembly
- [Mozilla Developer Network Documentation](https://developer.mozilla.org/en-US/docs/WebAssembly) - JavaScript side of things
- [WebAssembly Binary Toolkit](https://github.com/WebAssembly/binary-toolkit) - command-line assembler and disassembler
- [WebAssembly Toolkit for VSCode](https://github.com/WebAssembly/toolkit) - assembly code highlighter and assembler
- [Rust](https://rust-lang.org) - has built-in support for WebAssembly
- [Emscripten](https://emscripten.org) - LLVM bitcode to WebAssembly
- [Awesome Wasm Languages](https://github.com/sindresorhus/awesome-wasm-languages) - .NET, Elixir, Go, Java, Python, et al.
- [Awesome Wasm](https://github.com/sindresorhus/awesome-wasm) - curated list of awesome WebAssembly things



**[github.com/guyroyse/intro-to-webassembly](https://github.com/guyroyse/intro-to-webassembly)**

**[github.com/guyroyse/wasm-fizzbuzz](https://github.com/guyroyse/wasm-fizzbuzz)**



# Questions?



# Guy Royse

Developer Evangelist  
DataRobot

 @guyroyse

 [github.com/guyroyse](https://github.com/guyroyse)



