Catherine Larson, Danielle Carrol, Hayden Zhang

Cyber Programming Language

- History
- Language Goals
- Binding Choices
- Data Types
- Control Statements
- Memory Management
- Abstraction
- Modern Applications

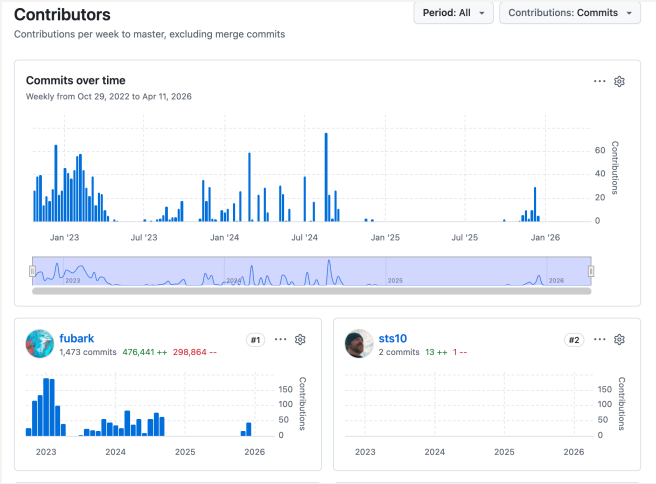
LANGUAGE HISTORY

A relatively new language!

- First release/deployment was in December 2023.
- Written primarily in Zig, though also utilizes C for compilation purposes
- BUT you cannot write C or Zig code directly in the Cyber playground – syntactical differences!

Who Is Fubark?

- A solo developer, not an anonymous collective (i.e., developed largely as a personal project).
- Ran into limitations with existing scripting languages and created Cyber to solve them.



LANGUAGE GOALS

Main Goal: Create a *safe*, *memory-efficient* language that has a *fast* runtime (that is similar to C) while mimicking Python's readability.

Subgoal I: Memory Efficiency

- Does NOT utilize garbage collection – rather, reference counts!

Subgoal 2: Fast Runtime

- Takes advantage of static typing for certain data structures.

Subgoal 3: Interoperability

- Ability to utilize the libraries from C and Zig. Targets ML libraries and game engines.

Subgoal 4: Concurrency

- Cyber allows for simultaneous thread execution.

DEEPER DIVE: STATIC TYPE BINDING

Key Aspects:

- Static typing allows for integrated safety!
- Variables MUST be declared with a value (i.e., not like Java!)
- Similar to SILLY – have to declare a variable with a keyword (var)
- Type is inferred upon declaration, but can optionally be specified.
- Once initialized, datatype for variable *cannot* be changed later!

Expected Output:

```
2
2
foo
2
133
```

```
var a int = 133

func example():
    var a = 2
    print(a)

    if (1 < 2):
        print(a)
        var a = "foo"
        print(a)

    print(a)

example()
print(a)
```

DATA TYPES

Primitive/ Built-in Types:

- Bool- true or false
- Int- 64-bit integer, can be stored as decimal, hex, octal or binary
- Float- all decimal and scientific notation values
- Str- immutable UTF-8 string; concatenate with + or the concat method

Custom/ Composite Types:

- struct – value type with named fields; stored flat in memory
- enum – a named set of constants; statically typed
- option (?) – a type that can hold a value or none
- List / Map – dynamic collections; reference types managed by ARC

CONTROL STATEMENTS

Conditional If/Else:

- Indentation-based blocks; no parentheses required around the condition.
- Else with a condition handles additional branches; else alone is the fallback.
- Variables used across branches must be declared before the if block to avoid scope issues.

```
var score = 77
var grade = ""
if (score > 90):
    grade = "A"
else (score > 80):
    grade = "B"
else:
    grade = "C"
print(grade)
```

```
C
```

CONTROL STATEMENTS

Loops – For / While :

- While loops on a condition.
- For loop syntax uses an arrow, then an alias for the index name.

For Loop Example:

```
var items = {1, 2, 3, 4, 5}
for items -> it:
   print(it) -- iterate list
```

```
1
2
3
4
5
```

While Loop Example:

```
var x = 0
while (x < 5):
   x = x+1
   print(x)
```

```
1
2
3
4
5
```

CONTROL STATEMENTS

Switch/ Case Matching:

- Matches values or enum variants.
- Can return a value directly.
- Else acts as the default case.

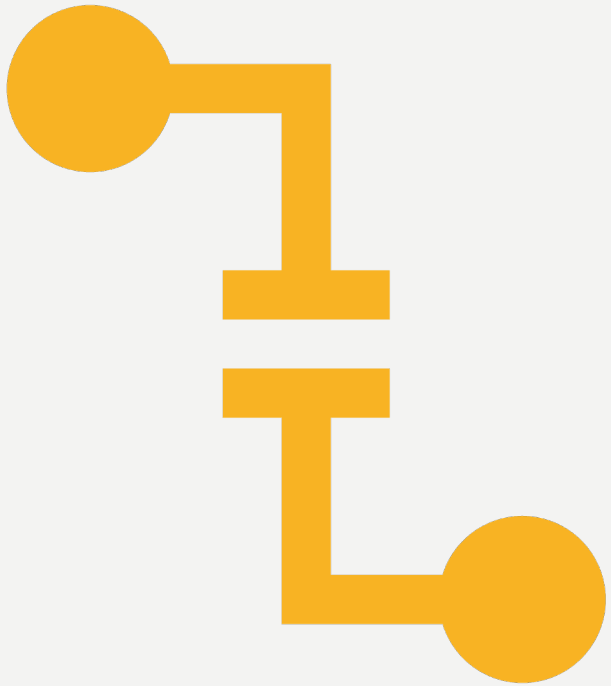
```
var color = 'green'
var action = switch (color):
    case 'green' => 'go'
    case 'red' => 'stop'
    else => none
```

```
go
```

MEMORY MANAGEMENT

Main Purpose

- Automatic Reference Counting ensures deterministic memory deallocation.
- Cycle collector handles the reference cycle, preventing memory leaks.
- Low-level memory manipulation when needed, just as C does.
  - & operator retrieves address.
  - * operator dereferences address.



MEMORY MANAGEMENT

Code Example

```
a:= 123
ref := &a
ref.* = 234
print(a)

inc(&a)
print(a)

fn inc(a &int):
      a.* = a.* + 1
```

```
234
235
```

ABSTRACTION

Core Abstractions of Cyber

- Cyber treats functions as primary values, which allows for high level programming patterns
- It features fiber multitasking which avoids overhead and complexity of OS level threads with lightweight concurrency
- It simplifies the difficulty of C interop, which allows for seamless integration with C and Zig libraries

ABSTRACTION

Code Example

```
type DataBuffer object:
      items any
func createBuffer() any:
       var b = DataBuffer{ items :
         [1, 2, 3, 4, 5] }
       print "Buffer allocated"
   print "Processing data..."
       print b.items.len()
       b.items = none
       print "Memory released."

var myBuffer = createBuffer()
```

```
Buffer allocated
Processing data...
5
Memory released.
```

CONCURRENCY

Utilizes threads!

- Threads may interact with each other through messaging (i.e., are *sendable*), or share memory.
- Threads defined with the keyword *spawn*.
- Can define subthreads, which are called *fibers* – these share memory with the parent queue but define their own stacks.
- The example below shares how to define a fiber, modified from example code within the documentation!

```
var task = spawn(fib, {40})
func fib(n int)int:
   if (n < 2):
      return n
   return fib(n-1)+ fib(n-2)
```

MODERN APPLICATIONS

Theoretical Possibilities

- Cyber is optimized for applications like game engines and robotics control logic where data on real time performance are critical.
- WebAssembly is another application that fubark claims uses cyber.
  - In theory, Cyber helps with the high-performance browser execution and run logic speed
- Fubark claims that Systems and Edge Computing use Cyber due to the environment requiring fast startup times and minimal memory footprints.

Disclaimer

- This is a new language so not many firms are using it!
- We couldn't find any evidence of real-world use cases of Cyber in industry.
- No sources exist to validate the claims that fubark makes.

REFERENCES

- Language Documentation
- Language GitHub Site
- Review of Language
- Discourse Surrounding Language
- Language Discord Server