# NIM

a whirlwind tour of the programming language (0.18)

xander johnson 2018

@metasyn

# CONTENTS

- overview
- motivation
- history
- features
- comparisons
- demo

# FEATURES

- efficient, expressive, elegant
- clean intuitive syntax
- high performance, compiled
- statically & strongly typed
- depedency free binaries
- cross platform

# MOTIVATION

- C++ speed
- scala-esque flexibility
- pythonic ergonomics
- GC'd a la golang
- "safe by default" like rust

# HISTORY

- 2005, MIT 2008 - Andreas Rumpf
- ~370 contributors, 5.2k stars
- Influences (in order of impact):
  - Modula 3, Delphi, Ada, C++, Python, Lisp, Oberon.
- Version 0.18
  - Prepping for 1.0 release

# FEATURES

- off-side (python)
- generics (C++)
- concepts (haskell's type class)
- metaprogramming (lisp)
- overloading (C)
- excellent C/C++ FFI

# (CROSS) COMPILE

- C
- C++
- Objective-C
- Javascript
- WASM (via emcscripten)

# SYNTAX

```
import strformat
type
  Person = object
    name*: string # Field is exported using `*`.
    age: Natural  # Natural type ensures the age is positive.

var people = [
  Person(name: "John", age: 45),
  Person(name: "Kate", age: 30)
]

for person in people:
  # Type-safe string interpolation.
  echo(fmt"{person.name} is {person.age} years old")
```

# CONTROVERSIAL

- case & style insensitivities

```
person.say_hello()
person.sayHello()
```

- uniform function call syntax (UFCS)
  - Rust, D

```
bark(dog)
dog.bark()
```

```nim
import sequtils

type Point = tuple[x, y: float]

proc `+`(p, q: Point): Point = (p.x + q.x, p.y + q.y)

proc `/`(p: Point, k: float): Point = (p.x / k, p.y / k)

proc average(points: seq[Point]): Point =
  foldl(points, a + b) / float(points.len)
```

- inlined at compile head
- zero overhead vs imperative loop

# FFI

```nim
proc printf(formatstr: cstring)
  {.header: "<stdio.h>", varargs.}
printf("%s %d\n", "foo", 5)
```

# MEMORY MANAGEMENT

- garbage collection by default
- (soft) realtime support
- manual options

- default
  - Deferred Reference Counting
  - Stack references aren't counted
  - Cycle detection via mark & sweep over full thread-local heap
- soft
  - max pause & step length
- manual
  - manual reference marking
  - (de)alloc, (de)allocShared, (de)allocCStringArray

# FLEXIBLE

```
import sequtils, future, strutils
let list = @["Zidong Yang", "Toufic Boubez"]

# Functional
list.map(
 (x: string) -> (string, string) => (x.split[0], x.split[1])
).echo


# Procedural
for name in list:
  echo((name.split[0], name.split[1]))
```

# TEMPLATES

```
template times(x: expr, y: stmt): stmt =
  for i in 1..x:
    y

10.times:
  echo "Hello World"
```

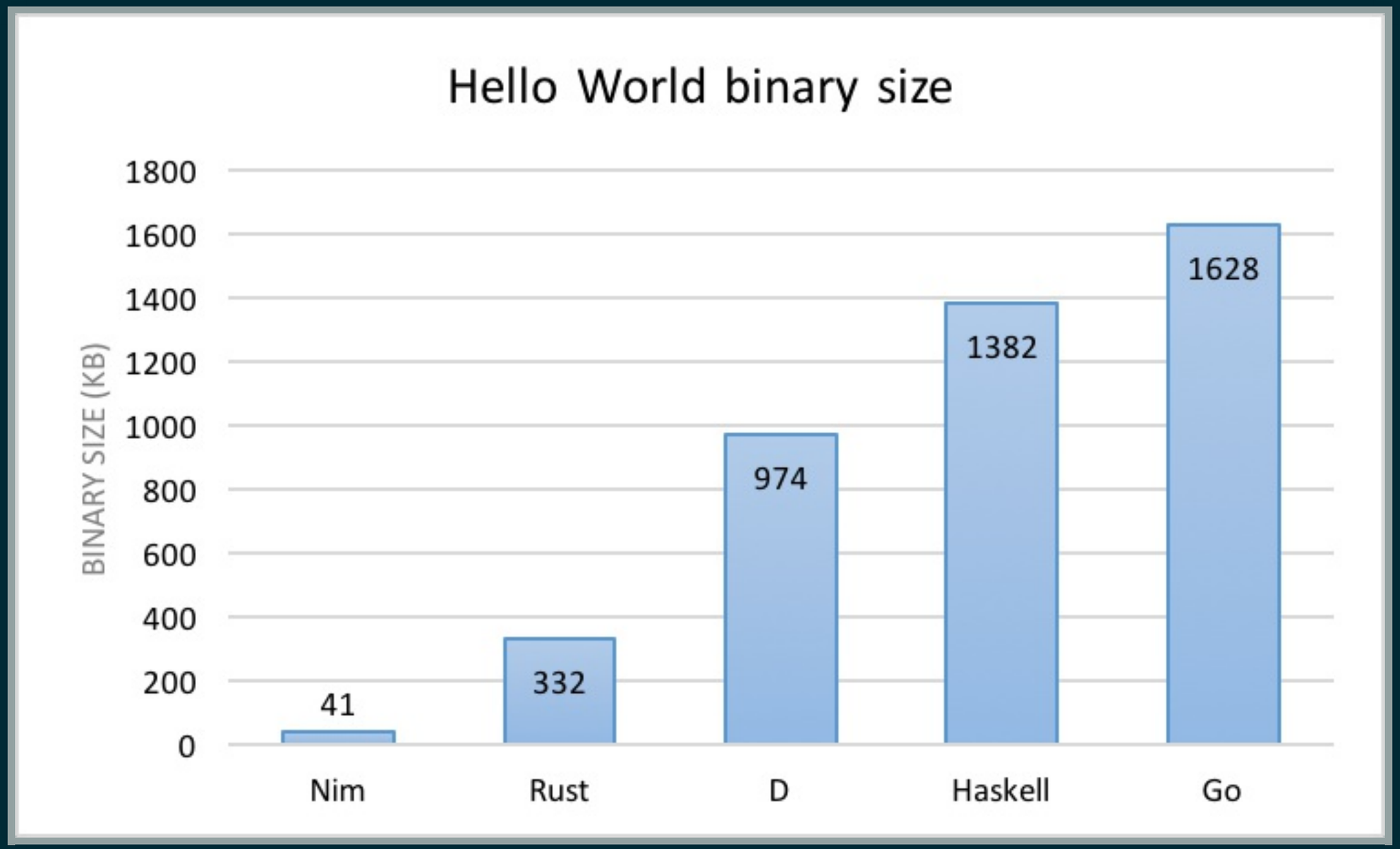
# OOP

```
type Animal = ref object of RootObj
  name: string
  age: int
method vocalize(this: Animal): string {.base.} = "..."
method ageHumanYrs(this: Animal): int {.base.} = this.age

type Dog = ref object of Animal
method vocalize(this: Dog): string = "woof"
method ageHumanYrs(this: Dog): int = this.age * 7

type Cat = ref object of Animal
method vocalize(this: Cat): string = "meow"
```

# DEPENDENCY FREE BINARIES



# SPEED

# PYTHON

```
BIG_NUMBER = 2 ** 24

def func_python(N):
    d = 0.0
    for i in range(N):
        d += (i % 3 - 1) * i
    return d
```

- 3.08 s

# CYTHON (2.5X)

```
def func_cython(N):
    cdef float d = 0.0 # only change
    for i in range(N):
        d += (i % 3 - 1) * i
    return d
```

- 1.25 s

# NIM (AS PYTHON PACKAGE)

```
nimble install nimpy
```

```
import nimpy
from math import `mod`

proc func_nim*(N: int): float {.exportpy.} =
  for i in 0..N:
    result += float((i mod 3 - 1) * i)
```

```
nim c -d:release --app:lib --out:loop.so loop.nim
```

# NIM-PYTHON (5X)

```
from loop import func_nim
func_nim(BIG_NUMBER)
```

- 608 ms

# NIM STANDALONE (94X)

```
import loop
from math import pow

const BIG_NUMBER = int(pow(2.0, 24.0))
echo func_nim(BIG_NUMBER)
```

```
nim c -d:release test.nim
time ./test
```

- 32 ms

# FORTRAN (AS PYTHON PACKAGE) (115X)

- fortran magic + f2py

```
subroutine func_fort(n, d)
      integer, intent(in) :: n
      double precision, intent(out) :: d
      integer :: i
      d = 0
      do i = 0, n - 1
        d = d + (mod(i, 3) - 1) * i
      end do
end subroutine func_fort
```

```
# python
func_fort(BIG_NUMBER)
```

- 26.7 ms

# DEMO

# MISSING

- corporate sponsor
- missing libraries
- no REPL / sorta

# FIN