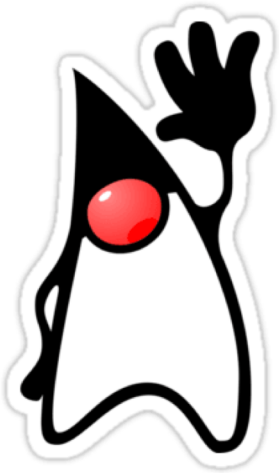
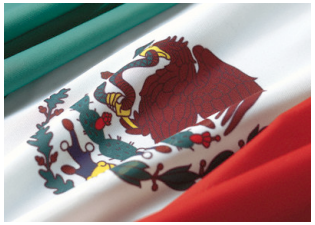


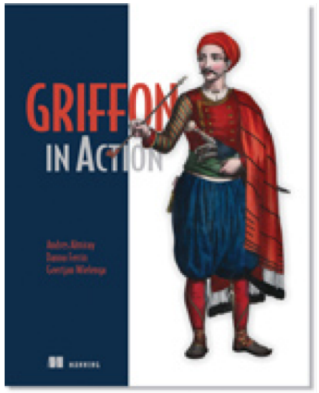
GO JAVA, GO!

ANDRES ALMIRAY

@AALMIRAY

ANDRESALMIRAY.COM



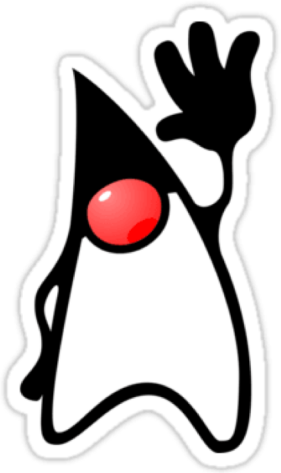


JAVA™
CHAMPIONS

ORACLE
Developer
Champion

HACKERGARTEN

@aalmiray



23

8

GO: GETTING STARTED

Official (executable) documentation

https://gobyexample.com/

Test your knowledge with

https://github.com/cdarwin/go-koans

HELLO WORLD (JAVA)

```
package main;

public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

$ java main/HelloWorld.java

$ javac –d classes main/HelloWorld.java

$ java –cp classes main.HelloWorld

HELLO WORLD (GO)

```
package main
import "fmt"

func main() {
    fmt.Println("Hello World")
}
```

$ go run hello-world.go

$ go build hello-world.go

$ ./hello-world

FAMILIAR FEATURES

VALUES (JAVA)

```
package main;

public class Values {
    public static void main(String[] args) {
        System.out.println("go" + "lang");

        System.out.println("1+1 = "+ (1+1));
        System.out.println("7.0/3.0 = " + (7.0/3.0));

        System.out.println(true && false);
        System.out.println(true || false);
        System.out.println(!true);
    }
}
```

VALUES (GO)

```
package main
import "fmt"

func main() {
    fmt.Println("go" + "lang")

    fmt.Println("1+1 = ", 1+1)
    fmt.Println("7.0/3.0 = ", 7.0/3.0)

    fmt.Println(true && false)
    fmt.Println(true || false)
    fmt.Println(!true)
}
```

VALUES (OUTPUT)

```
golang
1+1 = 2
7.0/3.0 = 2.3333333333333335
false
true
false
```

CONDITIONS (JAVA)

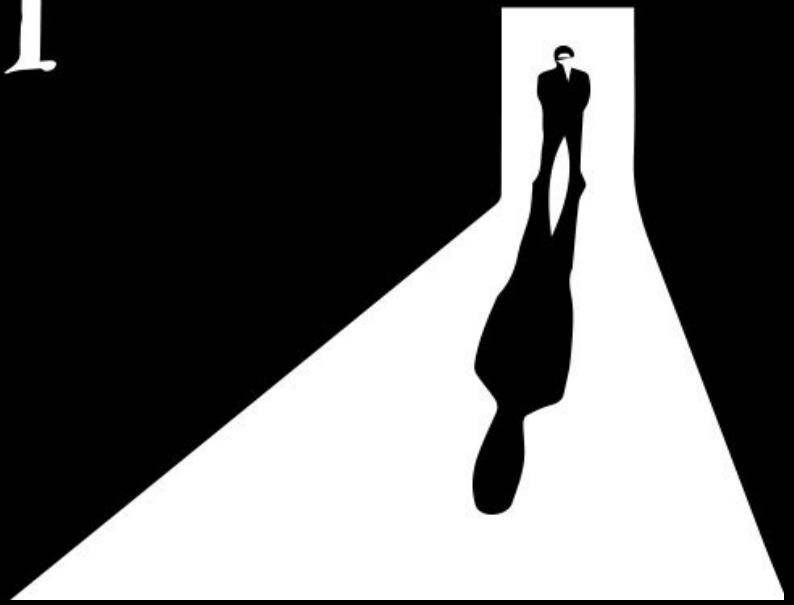
```
package main;

public class Conditions {
  public static void main(String[] args) {
    if (7%2 == 0) {
      System.out.println("7 is odd");
    } else {
      System.out.println("7 is even");
    }
  }
}
```

CONDITIONS (GO)

```
package main
import "fmt"

func main() {
    if 7%2 == 0 {
        fmt.Println("7 is odd")
    } else {
        fmt.Println("7 is even")
    }
}
```


"That's all Folks!"

The Twilight Zone



VISIBILITY

- There are 4 visibility modifiers in Java:
  - public, protected, private, and package private
  - Well… technically 5 -> modules
- There is a case convention in Go
  - Symbols starting with uppercase as public
  - Symbols starting with lowercase are private
  - That's it, no more, move along.

TYPE INFERENCE (JAVA)

- We've got verbosity reduction with the <> operator (JDK 7)

```
List<String> strings = new ArrayList<>();
```

- Next we've got type inference for local variables (JDK 10)

```
var strings = new ArrayList<String>();
```

- Use `var` in lambda expression arguments (JDK 11)

TYPE INFERENCE (GO)

- You may define and assign variables in this way

```
var strings = []string{"a","b","c"};
```

- Or use the short notation

```
strings := []string{"a","b","c"};
```

COLLECTIONS

- Slices and Maps (collections)

```
var sliceOfStrings = []string{"a", "b", "c"}

mapOfValues := make(map[string]int)
mapOfValues["foo"] = 1

anotherMap := map[string]int{"foo": 1}
```

ARRAYS

- Arrays look like slices but their length is part of the type

```
var an_array [5]int
another_one := [5]int{1,2,3,4,5}
```

- Any function that takes `[5]int` can't take `[4]int` or any other array with a different length than `5`.

FUNCTIONS

- Functions may have zero or more arguments
- Return type is defined after the argument list
- Symbol naming convention applies

```
func fib(n int) int {
    if n <= 1 {
        return n
    }
    return fib(n-1) + fib(n-2)
}
```

MULTIPLE RETURN VALUES

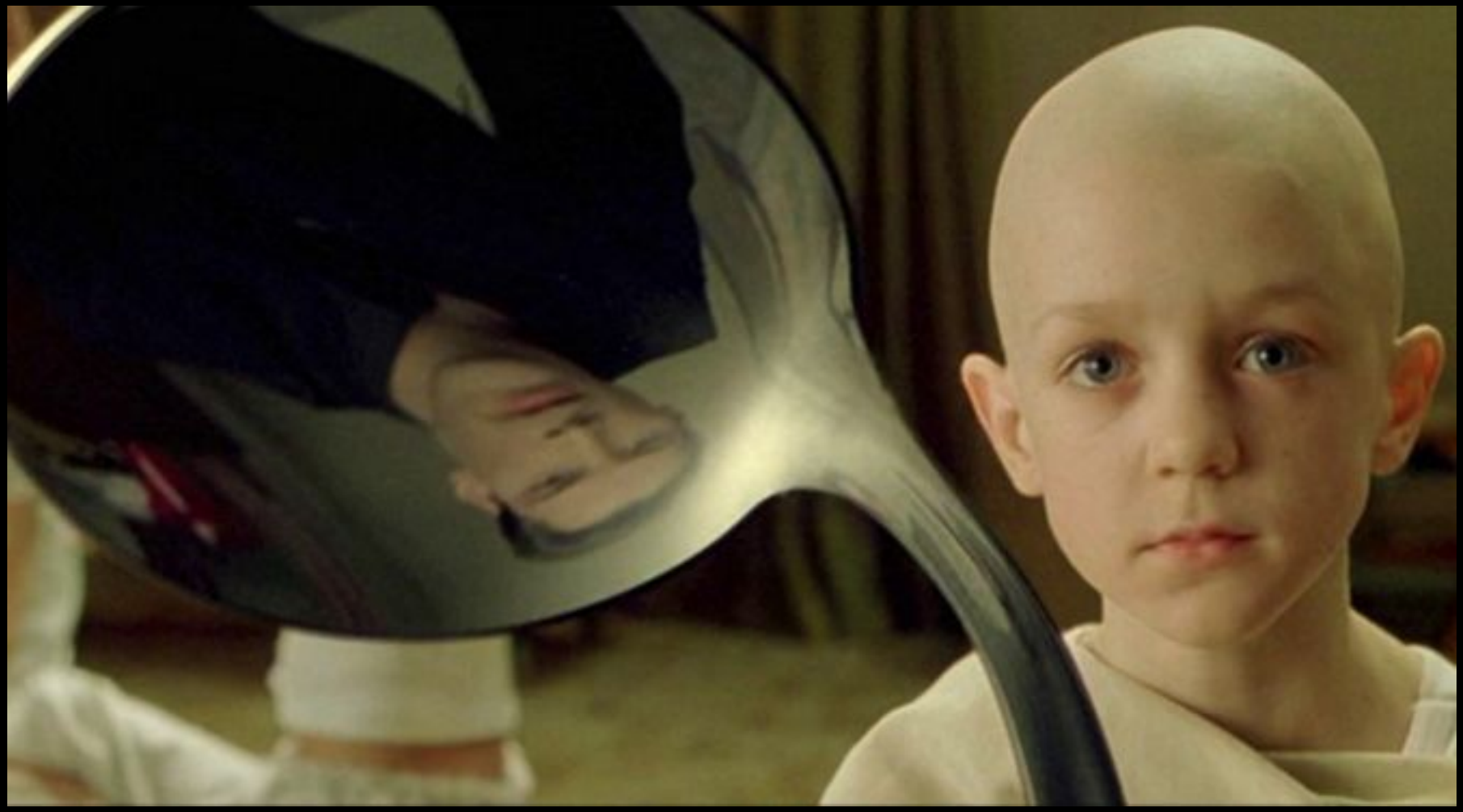
- Return as many values as needed

```
func thisAndTheOtherThing() (int,string) {
    // do some work
    return 0, "OK"
}
```

FUNCTIONS AS CODE

- Just like lambda expressions

```
package main
import "fmt"

func greeting_gen() func(string) string {
    return func(s string) string {
        return "Hello " + s
    }
}

func main() {
    fmt.Println(greeting_gen()("Go"))
}
```



THERE IS NO CLASS

BUT THERE IS STRUCT

- There's no equivalent to POJOs in Go
- You may create new types by leveraging structs

```
type Person struct {
    name string
    age int
}
```

CONSTRUCTORS

```
package main
import "fmt"

type Person struct {
    name string
    age int
}

func main() {
    p1 := Person{"Duke", 23}
    p2 := Person{name: "Duke", age: 23}
}
```


Gotcha!

SPOT THE COMPILE ERROR

```
package main
import "fmt"

type Person struct {
    name string
    age int
}

func main() {
    p1 := Person{"Duke", 23}
}
```

SPOT THE COMPILE ERROR

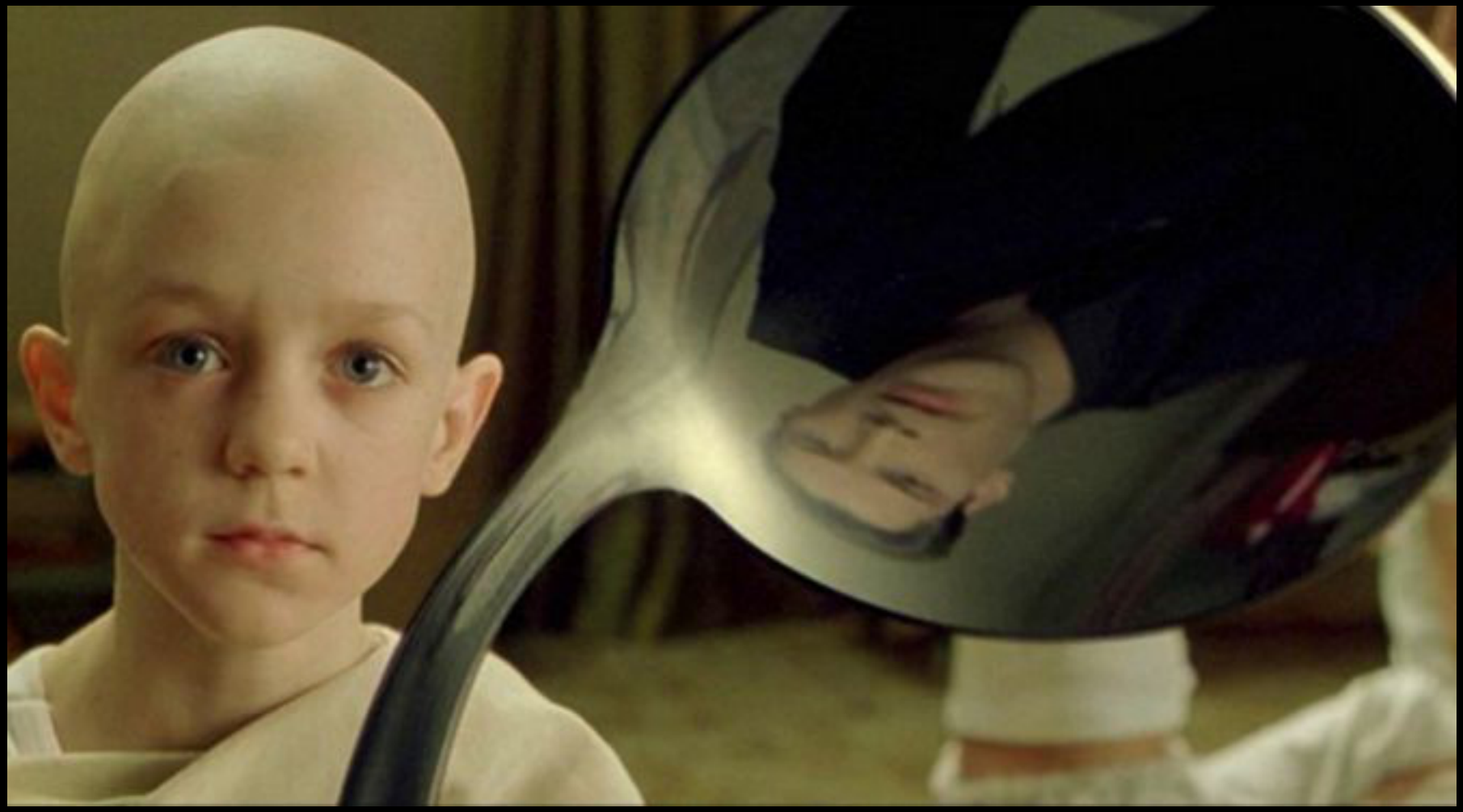
```
package main
import "fmt" // UNUSED!

type Person struct {
    name string
    age int
}

func main() {
    p1 := Person{"Duke", 23}  // UNUSED!
}
```

ONE SYNTAX TO RULE THEM ALL

GOFMT

- Everyone's Go code looks exactly the same.
- Tabs vs. Spaces? No more.
- Enables automated source code migration.

- https://golang.org/cmd/gofmt



THERE ARE NO METHODS

ATTACH FUNCTIONS TO TYPES

```
package main
import "fmt"

type Person struct {
    name string
    age int
}

func (p *Person) printAge() {
    fmt.Println("Age is = ", p.age)
}

func main() {
    p1 := Person{"Duke", 23}
    p1.printAge()
}
```

WHILE LOOP (JAVA)

```
package main;

public class While {
  public static void main(String[] args) {
    int i = 0;
    while(i <= 3) {
      System.out.println(i);
      i++;
    }
  }
}
```

WHILE LOOP (GO)

```
package main
import "fmt"

func main() {
    i := 0
    for i <= 3 {
        fmt.Println(i)
        i++
    }
}
```

FOR LOOP (JAVA)

```
package main;


public class For {
  public static void main(String[] args) {
    for(int i = 0; i <=3; i++) {
      System.out.println(i);
    }
  }
}
```

FOR LOOP (GO)

```
package main
import "fmt"

func main() {
    for i:= 0; i <= 3; i++ {
        fmt.Println(i)
    }
}
```

INFINITE LOOPS (JAVA)

```
package main;

public class Infinite {
  public static void main(String[] args) {
    while(true) {
      System.out.println("Infinite loop");
      break;
    }
  }
}
```

INFINITE LOOPS (JAVA)

```
package main;

public class Infinite {
  public static void main(String[] args) {
    for(;;) {
      System.out.println("Infinite loop");
      break;
    }
  }
}
```

INFINITE LOOPS (JAVA)

```
package main;

public class Infinite {
  public static void main(String[] args) {
    do {
      System.out.println("Infinite loop");
      break;
    } while(true);
  }
}
```

INFINITE LOOPS (GO)

```
package main
import "fmt"

func main() {
    for {
        fmt.Println("Infinite loop")
        break
    }
}
```


WEIRD
SCIENCE

INTERFACES

- Interfaces are implemented automatically as long as the type matches all methods.

- The `interface{}` type is roughly equivalent to `java.lang.Object`

```
package main

import "fmt"
import "math"

type geometry interface {
    area() float64
    perim() float64
}

type rect struct {
    width, height float64
}
type circle struct {
    radius float64
}
```

```
func (r rect) area() float64 {
    return r.width * r.height
}
func (r rect) perim() float64 {
    return 2*r.width + 2*r.height
}

func (c circle) area() float64 {
    return math.Pi * c.radius * c.radius
}
func (c circle) perim() float64 {
    return 2 * math.Pi * c.radius
}
```

```
func measure(g geometry) {
    fmt.Println(g)
    fmt.Println(g.area())
    fmt.Println(g.perim())
}

func main() {
    r := rect{width: 3, height: 4}
    c := circle{radius: 5}

    measure(r)
    measure(c)
}
```

ERRORS (1)

- Go doesn't have exceptions like Java does
- Errors are just another type that can be handled
- Use the multiple return feature to "throw" errors

```
func f1(arg int) (int, error) {
    if arg == 42 {
        return -1, errors.New("can't work with 42")
    }

    return arg + 3, nil
}
```

ERRORS (2)

- Handle, rethrow, or ignore

```
idx1, err := f1(42)
if err != nil {
    // handle or rethrow
}


idx2, _ := f1(42)
```


Gotcha!

TYPE CLONE VS TYPE ALIAS

- Type cloning

```
type foo int
```

- Type aliasing

```
type bar = int
```

- Instances of `foo` behave like `int` BUT they are not the same as `int`, that is, a method taking an `int` as argument can't take a `foo`.
- Instances of `bar` are identical to `int`, that is, anywhere an `int` fits so does a `bar` and viceversa.

OK,
BUT WHY?

HELLO WORLD (JAVA)

```
$ time java main/HelloWorld.java
Hello World

real    0m0.538s
user    0m0.918s
sys     0m0.069s

$ javac -d classes main/HelloWorld.java

$ time java -cp classes main.HelloWorld
Hello World

real    0m0.112s
user    0m0.104s
sys     0m0.029s
```

HELLO WORLD (GO)

```
$ time go run hello-world.go
Hello World

real    0m0.191s
user    0m0.135s
sys     0m0.080s

$ go build hello-world.go

$ time ./hello-world
Hello World

real    0m0.006s
user    0m0.001s
sys     0m0.004s
```

TIMES

	Java	Go	Percentage
Run			
Real	0.538	0.191	65.59
User	0.918	0.135	85.29
Sys	0.069	0.080	-15.94
Compile & Run			
Real	0.112	0.006	94.64
User	0.104	0.001	99.03
Sys	0.029	0.004	86.20

CROSS COMPILATION

- It's possible to compile binaries for different platforms at a single location

- https://dave.cheney.net/2015/03/03/cross-compilation-just-got-a-whole-lot-better-in-go-1-5

```
$ GOOS=darwin GOARCH=386 go build test.go
$ GOOS=linux GOARCH=arm GOARM=5 go build test.go
```


GRPC

HTTPS://GRPC.IO

gRPC is a modern, open source, high-performance remote procedure call (RPC) framework that can run anywhere. It enables client and server applications to communicate transparently, and makes it easier to build connected systems.

Stream data between client and server, in either direction, event both directions at the same time.

```proto
syntax = "proto3";

package org.kordamp.grpc.demo;
option java_multiple_files = true;

service HelloService {
    rpc helloUnary (HelloRequest) returns (HelloResponse);

    rpc helloServerStream (HelloRequest) returns (stream HelloResponse);

    rpc helloClientStream (stream HelloRequest) returns (HelloResponse);

    rpc helloBidirectional (stream HelloRequest) returns (stream HelloResponse);
}

message HelloRequest {
    string name = 1;
}

message HelloResponse {
    string reply = 1;
}
```

```java
@Override
public void helloUnary(HelloRequest request,
                       StreamObserver<HelloResponse> responseObserver) {
    doWithObserver(responseObserver, observer -> {
        observer.onNext(asResponse( s: "Hello " + request.getName()));
    });
}

private static HelloResponse asResponse(String s) {
    return HelloResponse.newBuilder()
        .setReply(s)
        .build();
}

private <T> void doWithObserver(@Nonnull StreamObserver<T> observer,
                                @Nonnull Consumer<StreamObserver<T>> consumer) {
    try {
        consumer.accept(observer);
        observer.onCompleted();
    } catch (Exception e) {
        observer.onError(e);
    }
}
```

```java
public class HelloClientUnary {
    public static void main(String[] args) {
        HelloClientUnary client = new HelloClientUnary();
        System.out.println(client.sayHello( input: "World"));
    }

    private String sayHello(String input) {
        return blockingStub.helloUnary(HelloRequest.newBuilder()
            .setName(input)
            .build()).getReply();
    }

    private final HelloServiceGrpc.HelloServiceBlockingStub blockingStub;

    private HelloClientUnary() {
        ManagedChannel channel = ManagedChannelBuilder.forAddress( name: "localhost",  port: 5678)
            .usePlaintext()
            .build();

        blockingStub = HelloServiceGrpc.newBlockingStub(channel);
    }
}
```

Want to learn more?

Get started by learning concepts and doing our hello world quickstart in language of your choice.

GET STARTED

Or go straight to Quick Start in the language of your choice:

C++	Java	Python
Go	Ruby	C#
Node.js	Android Java	Objective-C
PHP	Dart	Web

@aalmiray

Want to learn more?

Get started by learning concepts and doing our hello world quickstart in language of your choice.

GET STARTED

Or go straight to Quick Start in the language of your choice:

C++	Java	Python
Go	Ruby	C#
Node.js	Android Java	Objective-C
PHP	Dart	Web


WA

HTTPS://WEBASSEMBLY.ORG/

WebAssembly (abbreviated *Wasm*) is a binary instruction format for a stack-based virtual machine. Wasm is designed as a portable target for compilation of high-level languages, enabling deployment on the web for client and server applications.

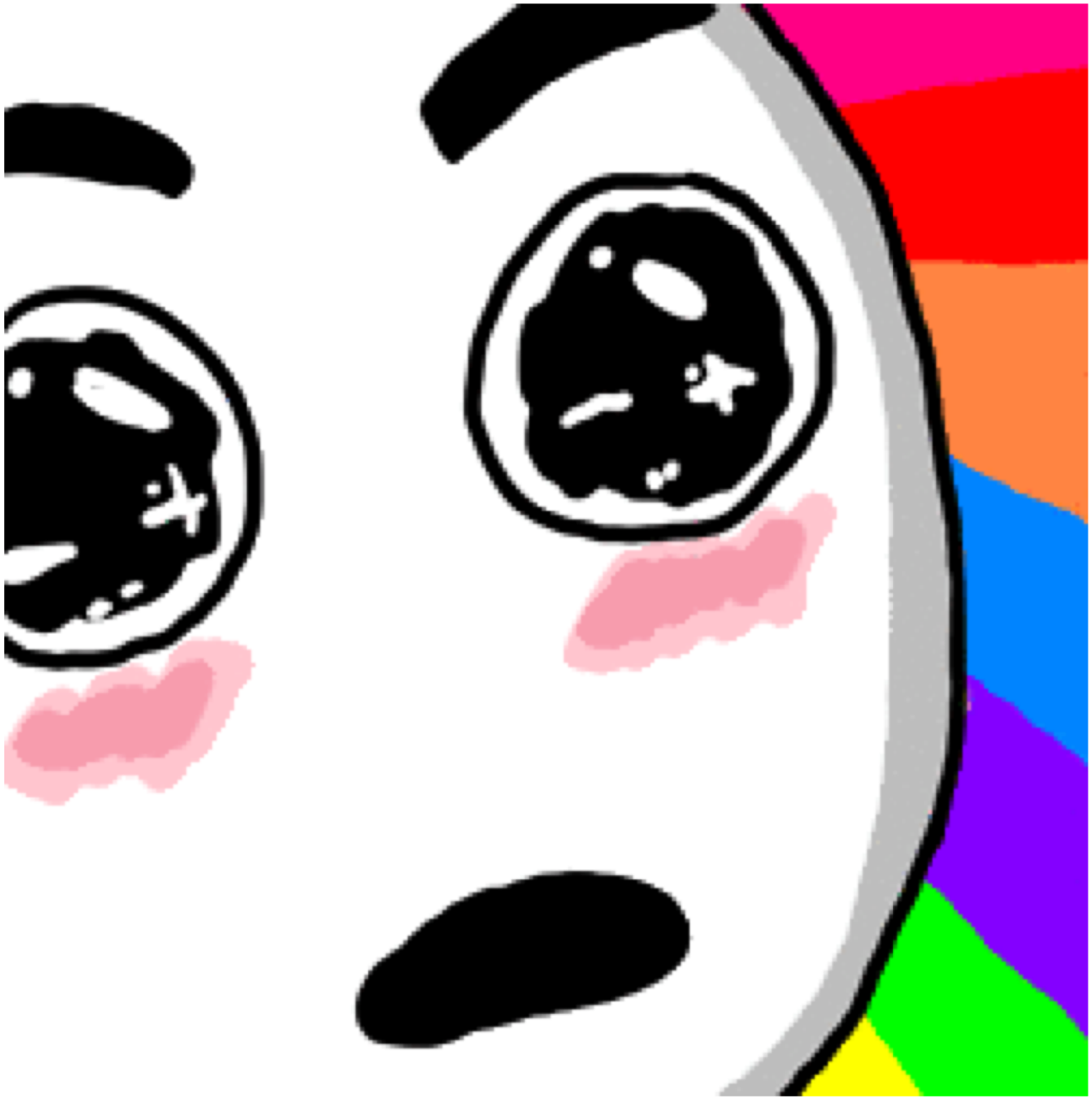
GO + WEBASSEMBLY

https://github.com/golang/go/wiki/WebAssembly

@aalmiray

HTTP://ANDRESALMIRAY.COM/NEWSLETTER

HTTP://ANDRESALMIRAY.COM/EDITORIAL



@aalmiray

THANK YOU!

ANDRES ALMIRAY

@AALMIRAY

ANDRESALMIRAY.COM