Go and Databases

An Introduction

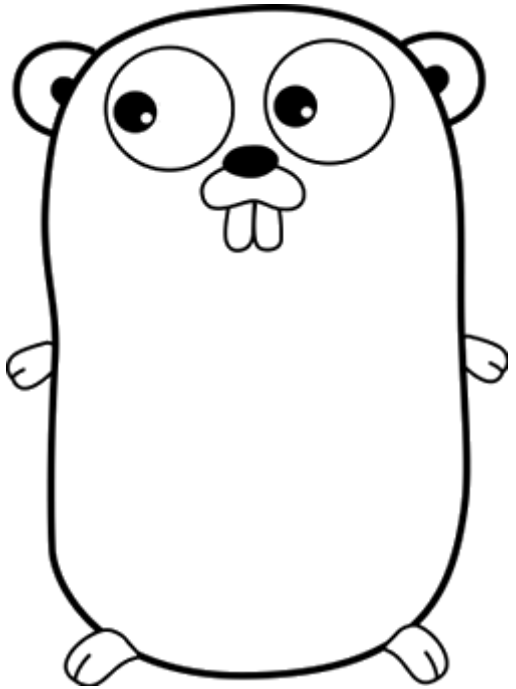
Hernán Rondelli
Universidad Nacional de General Sarmiento

bit.ly/go-and-db

What is Go?

What is Go?

- Created at Google by Ken Thompson, Rob Pike, and Robert Griesemer
- Compiled
- Garbage Collected



Gopher

C family

Go

```
    for i := 0; i < 10; i++ {
        fmt.Printf("%d\n", i)
    }
```

Run

C

```
    for (i = 0; i < 10; i++) {
        printf("%d\n", i);
    }
```

maaasoomeeeno...

Family tree

- ALGOL60
- Pascal → Modula-2 → Oberon → Oberon-2
- CPS → Squeak → Newsqueak → Alef
- C

"C + Pascal + Concurrency"

Hello, world!

Code

```
package main

import "fmt"

func main() {
    fmt.Printf("Hello, world!\n")
}
```

Run

Compile and Run

```
$ go run hello.go
```

Code formatting

```
$ gofmt -d hello.go
$ gofmt -w hello.go
```

Data Types

- Basic types: `int`, `float`, `complex`, `boolean`, `string`, `rune`
- Composite types: `arrays`, `slices`, `maps`, `structs`
- Reference types: `pointers`, `slices`, `maps`, `functions`
- Interface types

Basic Types

Declarations

```
var i int
var x float64
var z complex128
var ok bool
var s string
var c rune
```

Declarations with type inference

```
var i  = 0
var x  = 0.0
var z  = 0 + 0i

var (
    ok = true
    s  = "hello"
    c  = '😸'
)
```

Short Variable Declarations

```
i := 0
x := 0.0
z := 0 + 0i
ok := true
s := "hello"
c := '😸'
```

Composite types

Array

```
var a [5]int
b := [5]int{0, 2, 4, 6, 8}
```

Slice

```
var s0 []int
s1 := []int{1, 2, 3}
```

Map

```
var m map[string]int
```

...

Composite types: structs

privado

```
type punto struct {
    x, y int
}
```

Público

```
type Persona struct {
    Nombre   string
    Apellido string
    Edad     int
}
```

🤔

```
type Cuenta struct {
    númeroDeCuenta int
    monto          float64
}
```

Pointers

```
var i *int
j := new(int)

k := 3.14
p := &k
```

Functions

```
func Add(i, j int) int {
    return i + j
}

func AddAndSub(i, j int) (int, int) {
    return i + j, i - j
}

func AddSubAndDiv(i int, j int, k float64) (int, int, float64) {
    return i + j, i - j, float64(i) / k
}

func main() {
    a, b := AddAndSub(3, 5)
    fmt.Printf("%d %d\n", a, b)

    c, _ := AddAndSub(8, 2)
    fmt.Printf("%d\n", c)

    _, _, x := AddSubAndDiv(1, 3, 2)
    fmt.Printf("%f\n", x)
}
```

Run

if

```
package main

import "fmt"

func main() {
    n := 626
    if n == 626 {
        fmt.Printf("Viva Perón\n")
    } else {
        fmt.Printf("Aguante Cristina\n")
    }
}
```

Run

if with initialization statement

```
    a := 629
    if b := 626; a == b {
        fmt.Printf("Viva Perón\n")
    } else {
        fmt.Printf("Aguante Cristina\n")
    }
```

Run

for ...

C–like

```
package main

import (
    "fmt"
)

func main() {
    n := 10
    for i := 0; i < n; i++ {
        fmt.Printf("%d\n", i)
    }
}
```

Run

for ...

while-like

```
package main

import (
    "fmt"
)

func main() {
    n := 10

    i := 0
    for i < n {
        fmt.Printf("%d\n", i)
        i++
    }
}
```

Run

for range

for each–like

```
package main

import (
    "fmt"
)

func main() {
    a := [4]int{626, 629, 625, 624}

    for k, v := range a {
        fmt.Printf("%d %d\n", k, v)
    }
}
```

Run

Si se declara se usa

Error: Declared and not used

No se puede dejar cosas sin usar, si se declara se usa

```
package main

import (
    "fmt"
    "os"
)

func main() {
    a := 626
    fmt.Printf("Hello, world!\n")
}
```

Blank identifier

```
package main

import (
    "fmt"
    _ "os"
)

func main() {
    a := 626
    _ = a
    fmt.Printf("Hello, world!\n")
}
```

Run

Data input

```
package main

import (
    "fmt"
)

func main() {
    var nombre string

    fmt.Printf("Ingresá tu nombre: ")
    fmt.Scanf("%s", &nombre)

    fmt.Printf("Hola, %s!\n", nombre)
}
```

Time format

Magic date: 2006-01-02T15:04:05

```
package main

import (
    "fmt"
    "time"
)

func main() {
    fmt.Println(time.Now())
}
```

Run

```
    fmt.Println(time.Now().Format("02/01/2006"))
```

Run

```
    fmt.Println(time.Now().Format("hoy es 2 del 1 de 2006"))
```

Run

```
    fmt.Println(time.Now().Format("2006-01-02T15:04:05"))
```

Run

Time example

```
func main() {
    go hello5()
    go hello10()

    // var s string
    // fmt.Scanf("%s", s)
    time.Sleep(30 * time.Second)
}

func hello5() {
    for {
        fmt.Printf("%v: Viva Perón!\n", time.Now().Format("15:04:05"))
        time.Sleep(5 * time.Second)
    }
}

func hello10() {
    for {
        fmt.Printf("%v: Aguante Cristina!\n", time.Now().Format("15:04:05"))
        time.Sleep(10 * time.Second)
    }
}
```

Run

More info

- Get started with Go

golang.org/doc/tutorial/getting-started (https://golang.org/doc/tutorial/getting-started)

- Documentation

golang.org/doc (https://golang.org/doc)

Relational Databases in Go

Relational Databases in Go ...

Import declarations

```
package main

import (
    "database/sql"
    "fmt"
    _ "github.com/lib/pq"
    "log"
)

func main() {
    // acá va el código de mi "aplicacion-de-bases-de-datos"
}
```

... Relational Databases in Go ...

Open database connection

```
    db,err := sql.Open("postgres", "user=lucifer host=localhost dbname=postgres sslmode=disable")
    if err != nil {
        log.Fatal(err)
    }
    defer db.Close()
```

... Relational Databases in Go ...

Create table

```
    _, err = db.Exec(`create table alumne (legajo int, nombre text, apellido text)`)
    if err != nil {
        log.Fatal(err)
    }
```

Insert into

```
    _, err = db.Exec(`insert into alumne values (1, 'Cristina', 'Kirchner');
                      insert into alumne values (2, 'Juan Domingo', 'Perón');`)

    if err != nil {
        log.Fatal(err)
    }
```

... Relational Databases in Go

```
type alumne struct {
    legajo           int
    nombre, apellido string
}
```

Query

```
    rows, err := db.Query(`select * from alumne`)
    if err != nil {
        log.Fatal(err)
    }
    defer rows.Close()
    var a alumne
    for rows.Next() {
        if err := rows.Scan(&a.legajo, &a.nombre, &a.apellido); err != nil {
            log.Fatal(err)
        }
        fmt.Printf("%v %v %v\n", a.legajo, a.nombre, a.apellido)
    }
    if err = rows.Err(); err != nil {
        log.Fatal(err)
    }
```

Complete example ...

```
package main

import (
    "database/sql"
    "fmt"
    _ "github.com/lib/pq"
    "log"
)

type alumne struct {
    legajo           int
    nombre, apellido string
}
```

...

... Complete example ...

```
func createDatabase() {
    db,err := sql.Open("postgres", "user=lucifer host=localhost dbname=postgres sslmode=disable")
    if err != nil {
        log.Fatal(err)
    }
    defer db.Close()

    _, err = db.Exec(`create database guarani`)
    if err != nil {
        log.Fatal(err)
    }
}
```

...

... Complete example ...

```
func main() {
    createDatabase()

    db, err := sql.Open("postgres", "user=lucifer host=localhost dbname=guarani sslmode=disable")
    if err != nil {
        log.Fatal(err)
    }
    defer db.Close()

    _, err = db.Exec(`create table alumne (legajo int, nombre text, apellido text)`)
    if err != nil {
        log.Fatal(err)
    }

    _, err = db.Exec(`insert into alumne values (1, 'Cristina', 'Kirchner');
                      insert into alumne values (2, 'Juan Domingo', 'Perón');`)

    if err != nil {
        log.Fatal(err)
    }
```

...

... Complete example

```
    rows, err := db.Query(`select * from alumne`)
    if err != nil {
        log.Fatal(err)
    }
    defer rows.Close()
    var a alumne
    for rows.Next() {
        if err := rows.Scan(&a.legajo, &a.nombre, &a.apellido); err != nil {
            log.Fatal(err)
        }
        fmt.Printf("%v %v %v\n", a.legajo, a.nombre, a.apellido)
    }
    if err = rows.Err(); err != nil {
        log.Fatal(err)
    }
}
```

Relational Databases in Go

Creating module

```
$ mkdir aplicacion-de-bases-de-datos
$ cd aplicacion-de-bases-de-datos
$ go mod init aplicacion-de-bases-de-datos
go: creating new go.mod: module aplicacion-de-bases-de-datos
```

Writing code

```
$ vi main.go
```

Adding dependencies

```
$ go mod tidy
go: finding module for package github.com/lib/pq
go: found github.com/lib/pq in github.com/lib/pq v1.10.2
```

Running code

```
$ go run .
```

JSON

JSON

- JSON (JavaScript Object Notation)
- Structured information
- Standard, universal support
- Simple
- Strings, numbers, booleans, arrays, and objects

JSON data types to Go data types

- JSON numbers → Go int's and float's

```
626
-273.15
```

- JSON booleans → Go booleans

```
false
true
```

- JSON strings → Go strings

```
"Viva Perón"
```

- JSON arrays → Go arrays, and slices

```
["Viva", "Perón"]
```

JSON data types to Go data types

- JSON objects → Go structs

```
{
    "Title": "Casablanca",
    "released": 1942,
    "Actors": [
        "Humphrey Bogart",
        "Ingrid Bergman"
    ]
}
```

JSON Document

movies.json

```
[
    {
        "Title": "Casablanca",
        "released": 1942,
        "Actors": [
            "Humphrey Bogart",
            "Ingrid Bergman"
        ]
    },
    {
        "Title": "Lilo & Stitch",
        "released": 2002,
        "color": true,
        "Actors": [
            "Chris Sanders"
        ]
    }
]
```

Encoding and Decoding JSON

- Marshaling (encoding)

Go → JSON

- Unmarshaling (decoding)

JSON → Go

JSON example ...

```
package main

import (
    "encoding/json"
    "fmt"
    "log"
)

// Sólo se marshalean los fields públicos
type Movie struct {
    Title  string
    Year   int  `json:"released"`
    Color  bool `json:"color,omitempty"`
    Actors []string
}
```

...

... JSON example

```
func main() {
    movies := []Movie{
        {Title: "Casablanca", Year: 1942, Color: false,
            Actors: []string{"Humphrey Bogart", "Ingrid Bergman"}},
        {Title: "Lilo & Stitch", Year: 2002, Color: true,
            Actors: []string{"Chris Sanders"}},
    }
    fmt.Printf("%v\n", movies)

    data, err := json.MarshalIndent(movies, "", "    ") //data, err := json.Marshal(movies)
    if err != nil {
        log.Fatalf("JSON marshaling failed: %s", err)
    }
    fmt.Printf("%s\n", data)

    var películas []Movie
    err = json.Unmarshal(data, &películas)
    if err != nil {
        log.Fatalf("JSON unmarshaling failed: %s", err)
    }
    fmt.Printf("%v\n", películas)
}
```

Run

NoSQL Databases in Go

NoSQL Databases in Go (BoltDB)

- key/value store
- Embedded (no service)
- Database: **filename.db**
- Bucket: **key → value**

Buckets

key → value

alumne:

```
1 → {"legajo": 1, "nombre": "Cristina", "apellido": "Kirchner"}
2 → {"legajo": 2, "nombre": "Diego Armando", "apellido": "Maradona"}
3 → {"legajo": 3, "nombre": "Juan Carlos", "apellido": "Olave"}
```

BoltDB

Características

- Permite una única transacción de read/write (rw)
- Permite muchas transacciones read only (ro)

BoltDB

Import declarations

```
package main

import (
    "fmt"
    "log"
    bolt "go.etcd.io/bbolt"
)

func main() {
    db, err := bolt.Open("guaraní.db", 0600, nil)
    if err != nil {
        log.Fatal(err)
    }
    defer db.Close()

    //...
}
```

Write to buckets

```
func CreateUpdate(db *bolt.DB, bucketName string, key []byte, val []byte) error {
    // abre transacción de escritura
    tx, err := db.Begin(true)
    if err != nil {
        return err
    }
    defer tx.Rollback()

    b, _ := tx.CreateBucketIfNotExists([]byte(bucketName))

    err = b.Put(key, val)
    if err != nil {
        return err
    }

    // cierra transacción
    if err := tx.Commit(); err != nil {
        return err
    }

    return nil
}
```

Read from buckets

```
func ReadUnique(db *bolt.DB, bucketName string, key []byte) ([]byte, error) {
    var buf []byte

    // abre una transacción de lectura
    err := db.View(func(tx *bolt.Tx) error {
        b := tx.Bucket([]byte(bucketName))
        buf = b.Get(key)
        return nil
    })

    return buf, err
}
```

Full code ...

```
package main

import (
    "encoding/json"
    "fmt"
    bolt "go.etcd.io/bbolt"
    "log"
    "strconv"
)

type Alumne struct {
    Legajo   int
    Nombre   string
    Apellido string
}

//...
```

... Full code ...

```
func CreateUpdate(db *bolt.DB, bucketName string, key []byte, val []byte) error {
    // abre transacción de escritura
    tx, err := db.Begin(true)
    if err != nil {
        return err
    }
    defer tx.Rollback()

    b, _ := tx.CreateBucketIfNotExists([]byte(bucketName))

    err = b.Put(key, val)
    if err != nil {
        return err
    }

    // cierra transacción
    if err := tx.Commit(); err != nil {
        return err
    }

    return nil
}
```

... Full code ...

```
func ReadUnique(db *bolt.DB, bucketName string, key []byte) ([]byte, error) {
    var buf []byte

    // abre una transacción de lectura
    err := db.View(func(tx *bolt.Tx) error {
        b := tx.Bucket([]byte(bucketName))
        buf = b.Get(key)
        return nil
    })

    return buf, err
}
```

... Full code

```
//...

func main() {
    db, err := bolt.Open("guaraní.db", 0600, nil)
    if err != nil {
        log.Fatal(err)
    }
    defer db.Close()

    cristina := Alumne{1, "Cristina", "Kirchner"}
    data, err := json.Marshal(cristina)
    if err != nil {
        log.Fatal(err)
    }

    CreateUpdate(db, "alumne", []byte(strconv.Itoa(cristina.Legajo)), data)

    resultado, err := ReadUnique(db, "alumne", []byte(strconv.Itoa(cristina.Legajo)))

    fmt.Printf("%s\n", resultado)
}
```

NoSQL Databases in Go (BoltDB)

Creating module

```
$ mkdir sui-guarani
$ cd sui-guarani
$ go mod init sui-guarani
go: creating new go.mod: module sui-guarani
```

Writing code

```
$ vi app-boltdb.go
```

Adding dependencies

```
$ go mod tidy
go: finding module for package go.etcd.io/bbolt
go: found go.etcd.io/bbolt in go.etcd.io/bbolt v1.3.6
```

Running code

```
$ go run .
```

One more thing…

More info

- PostgreSQL documentation

www.postgresql.org/docs/current (https://www.postgresql.org/docs/current)

- BoltDB documentation

pkg.go.dev/go.etcd.io/bbolt (https://pkg.go.dev/go.etcd.io/bbolt)

Thank you

Hernán Rondelli
Universidad Nacional de General Sarmiento
lucifer.unix.cabj@gmail.com (mailto:lucifer.unix.cabj@gmail.com)