



# Programmation Comparée : GO-LANG

Université Paris VII M2 Informatique 2018  
Idir LANKRI | Krimo HAMADI | Yung-Kun HSIEH



# Plan

- Histoire de Go.
- Caractéristiques de Go.
- Écosystème de Go.
- Exemple de code 1 : La syntaxe.
- Exemple de code 2 : La gestion des erreurs.
- Exemple de code 3 : Tests unitaires.
- Exemple de code 4 : Goroutine.
- Modèle concurrent de Go.
- Comparaison de Goroutine et Thread
- Go et OCaml (Function closure, pattern matching)

# Histoire de Go

- Influencé par C, C++, Java, Python.
- Développé par Google en 2007 pour faciliter et améliorer les travaux chez Google comme une service de génie logiciel.
- “Le C du XXI<sup>e</sup> siècle”.

Mar 2018	Mar 2017	Change	Programming Language	Ratings	Change
1	1		Java	14.941%	-1.44%
2	2		C	12.760%	+5.02%
3	3		C++	6.452%	+1.27%
4	5	▲	Python	5.869%	+1.95%
5	4	▼	C#	5.067%	+0.66%
6	6		Visual Basic .NET	4.085%	+0.91%
7	7		PHP	4.010%	+1.00%
8	8		JavaScript	3.916%	+1.25%
9	12	▲	Ruby	2.744%	+0.49%
10	-	▲	SQL	2.686%	+2.69%
11	11		Perl	2.233%	-0.03%
12	10	▼	Swift	2.143%	-0.13%
13	9	▼	Delphi/Object Pascal	1.792%	-0.75%
14	16	▲	Objective-C	1.774%	-0.22%
15	15		Visual Basic	1.741%	-0.27%
16	13	▼	Assembly language	1.707%	-0.53%
17	17		Go	1.444%	-0.54%
18	18		MATLAB	1.408%	-0.45%
19	19		PL/SQL	1.327%	-0.34%
20	14	▼	R	1.128%	-0.89%

The TIOBE Programming Community Index



# Caractéristiques de Go

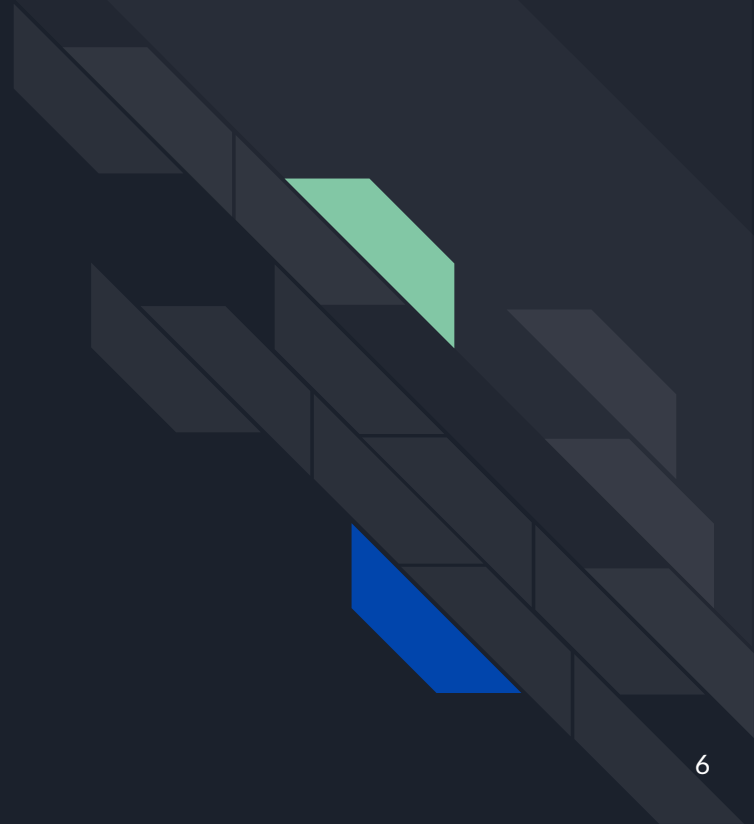
- Go est un langage de programmation compilé et concurrent.
- Langage impératif.
- Typage statique (à la compilation), fort (pas de conversion implicite), inféré (pas de type explicite) en partie.
- Garbage collection.

# Écosystème de Go

- Convention de nom de fichiers et répertoires.
- Configurer la variable d'environnement GOPATH.
  - `export GOPATH=/path/to/project...`
- Tests unitaires intégrés.
  - `# go test`
- Gestion de paquets.
  - `# go get`  
`http://github.com/golang/example/hello`

```
bin/
  hello                                # command executable
  outyet                               # command executable
pkg/
  linux_amd64/
    github.com/golang/example/
      stringutil.a                    # package object
src/
  github.com/golang/example/
    .git/                             # Git repository metadata
    hello/
      hello.go                        # command source
    outyet/
      main.go                         # command source
      main_test.go                   # test source
    stringutil/
      reverse.go                     # package source
      reverse_test.go                # test source
  golang.org/x/image/
    .git/                             # Git repository metadata
    bmp/
      reader.go                      # package source
      writer.go                      # package source
... (many more repositories and packages omitted) ...
```

# Examples



# Exemple 1 : La syntaxe (for, inférence des types, import)

## Spreadsheet.go

1. `func` nom\_de\_fonction (param type)  
type\_renvoye { ...  
}
2. nom\_de\_var := valeur
3. `var` nom\_de\_var type
4. `for` index, item := range list  
{ //pas d'accolades ici!!  
}
5. nom\_paquet.fonction

```
import (  
    "bufio"  
    "fmt"  
    "io"  
    "io/ioutil"  
    "log"  
    "os"  
    "parse"  
    "path/filepath"  
    "share"  
    "strconv"  
    "strings"  
    "sync"  
)
```

```
func Evaluate(bin_repo string, doWriteFinalValue bool) int {  
    files, err := ioutil.ReadDir(bin_repo + "/FORMULAS/")  
    if err != nil {  
        log.Fatal(err)  
    }  
  
    formula_list := make([]string, len(files))  
    for idx, f := range files {  
        formula_list[idx] = f.Name()  
    }  
  
    formula_list = SortByDependency(formula_list)  
    var wg sync.WaitGroup  
    for _, f := range formula_list {  
        wg.Add(1)  
        go EvaluateFormula(bin_repo, f, doWriteFinalValue, &wg)  
    }  
    wg.Wait()  
    return len(files)  
}
```

# Exemple 1 : La syntaxe ("receiver", structure)

Binary file.go

1. `func` (receiver) nom\_de\_fonction (param type) type\_renvoye { ... }
2. Composite literal : `nom_de_struct { ..., ... }`
3. Fonction publique / privée : `Close()` / `close()`
4. Envoyer une référence d'instance créé dans la fonction.

```
func newBinaryFile(file_path string) *BinFile {
    createFileIfNotExists(file_path)
    f, err_of := os.OpenFile(file_path, os.O_APPEND|os.O_CREATE|os.O_RDWR, 0644)
    if err_of != nil {
        log.Fatal(err_of)
        return nil
    }
    bin := BinFile{[]uint32{}, file_path, f, sync.Mutex{}}
    return &bin
}
```

```
type BinFile struct {
    data    []uint32
    file    string
    opened_f *os.File
    mut     sync.Mutex
}
```

```
func (b *BinFile) Close() error {
    b.mut.Lock()
    defer b.mut.Unlock()
    b.close()
    return nil
}
```

```
func (b *BinFile) close() {
    if b.opened_f != nil {
        err := b.opened_f.Close()
        if err != nil {
            panic(err)
        }
        b.opened_f = nil
    }
}
```



# Exemple 2 : La gestion des erreurs

## Binary file.go

1. “**defer** nom\_de\_fonction” permet d'exécuter nom\_de\_fonction à la fin de la fonction dans laquelle il est appelé.
2. Renvoyer l'erreur comme une valeur de retour.
3. Appelant doit toujours vérifier l'erreur -- il n'y pas de try {...} catch {...} en Go.
4. Terminer le program avec erreur : **panic**(error)
5. Si besoin, on peut continuer le program de **panic** par appel **recover()** dans la fonction de **defer**.

```
defer func() {  
    if r := recover(); r != nil {  
        fmt.Println("Recovered in f", r)  
    }  
}()
```

```
func (b *BinFile) ReadAll() ([]uint32, error) {  
    b.mut.Lock() //Ensure that no one is writing while reading  
    defer b.mut.Unlock()  
    f, err := os.OpenFile(b.file, os.O_APPEND|os.O_CREATE|os.O_RDONLY, 0644)  
    if err != nil {  
        panic(err)  
    }  
    defer f.Close()  
    br := bufio.NewReader(f)  
    for {  
        bytes := make([]byte, 4)  
        _, err := br.Read(bytes)  
        if err == io.EOF {  
            return b.data, nil  
        }  
        if err != nil {  
            return nil, err  
        }  
  
        d := binary.BigEndian.Uint32(bytes)  
        //fmt.Printf("Read %d bytes from file: %x\n", nbRead, d)  
        b.data = append(b.data, d)  
    }  
    return b.data, nil  
}
```

# Exemple 3 : Test unitaire

## Spreadsheet test.go

1. nom\_de\_fichier\_test.go
2. `func TestXXX (t *testing.T) { ... }`
3. `t.Fatal(message)` pour notifier d'erreur.
4. Go trouvera tous les tests automatiquement.
5. Type polymorphism : `interface {}`

```
== RUN    TestToFormula
--- PASS: TestToFormula (0.00s)
== RUN    TestBinFileToFormula
--- PASS: TestBinFileToFormula (0.00s)
== RUN    TestFromFile
2018/03/11 23:58:39 Purge directory: /var
2018/03/11 23:58:39 New directory created
2018/03/11 23:58:39 BinFileMgr initialed:
2018/03/11 23:58:39 New directory created
--- PASS: TestFromFile (0.00s)
== RUN    Test10LinesBigmamaFile
2018/03/11 23:58:39 Purge directory: /var
2018/03/11 23:58:39 New directory created
2018/03/11 23:58:39 New directory created
--- PASS: Test10LinesBigmamaFile (18.72s)
== RUN    TestEvaluate
--- PASS: TestEvaluate (0.00s)
== RUN    TestCounting
--- PASS: TestCounting (0.00s)
PASS
ok      spreadsheet    18.730s
```

```
package spreadsheet
```

```
import (
    "parse"
    "share"
    "testing"
)
```

```
func TestToFormula(t *testing.T) {
    f := ToFormula("#(0,0,50,50,1)")
    share.AssertEqual(t, f.xSource, uint32(0), "xSource is not uint32 0 from parsing formula string.")
    share.AssertEqual(t, f.ySource, uint32(0), "ySource is not uint32 0 from parsing formula string.")
    share.AssertEqual(t, f.xDestination, uint32(50), "xDestination is not uint32 50 from parsing formula string.")
    share.AssertEqual(t, f.yDestination, uint32(50), "yDestination is not uint32 50 from parsing formula string.")
    share.AssertEqual(t, f.value, uint32(1), "value is not uint32 1 from parsing formula string.")
}
```

```
func AssertEqual(t *testing.T, a interface{}, b interface{}, message string) {
    if reflect.DeepEqual(a, b) {
        return
    } else {
        message = fmt.Sprintf("%s: %v != %v", message, a, b)
    }
    t.Fatal(message)
}
```

# Exemple 4 : Goroutine

## Spreadsheet.go

1. `go` nom\_de\_fonction
2. `sync.WaitGroup`
3. `wg.Add(1)` avant `go` routine obligatoire.
4. `wg.Done()` quand le routine finit.
5. `wg.Wait()` pour attendre tous les routines finissent.

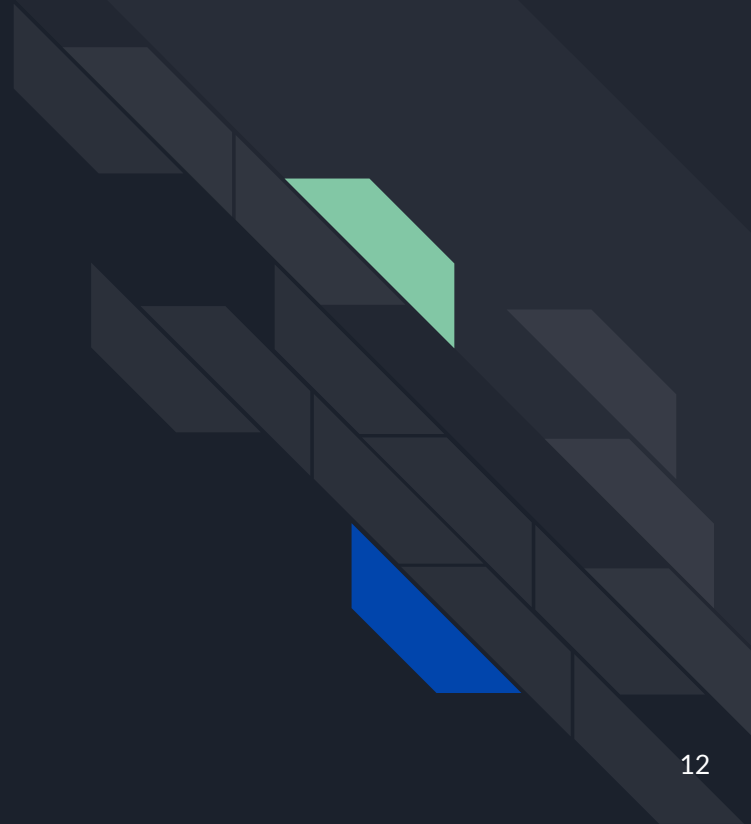
```
func Evaluate(bin_repo string, doWriteFinalValue bool) int {
    files, err := ioutil.ReadDir(bin_repo + "/FORMULAS/")
    if err != nil {
        log.Fatal(err)
    }

    formula_list := make([]string, len(files))
    for idx, f := range files {
        formula_list[idx] = f.Name()
    }

    formula_list = SortByDependency(formula_list)
    var wg sync.WaitGroup
    for _, f := range formula_list {
        wg.Add(1)
        go EvaluateFormula(bin_repo, f, doWriteFinalValue, &wg)
    }
    wg.Wait()
    return len(files)
}
```

```
func EvaluateFormula(bin_repo string, formulaName string, doWriteFinalValue bool, wg *sync.WaitGroup) uint32 {
    if wg != nil { //When WaitGroup is not provided, nothing to notify
        defer wg.Done()
    }
}
```

# Comparaisons





# Comparaison de Go et Java

- Les *goroutines* sont gérées par le runtime de Go.
  - Gestion de la mémoire allouée.
  - Ordonnancement.
- Plus légères et plus flexibles que les threads.



# Modèle concurrent de Go

- Plusieurs activités (*goroutines*) indépendantes s'exécutant de manière concurrente.
- Communication via des canaux (*channels*).
- Canaux typés
  - Type des messages fixé.
  - Bidirectionnel ou unidirectionnel.
- La concurrence selon Go :

*Do not communicate by sharing memory; instead, share memory by communicating.*

- Permet d'écrire des programmes concurrents plus fiables.



# Comparaison de Go et OCaml

- Fonctions de première classe.
- Interfaces permettent d'écrire du code générique.
- Types dynamiques
  - Permettent le polymorphisme ad hoc
  - Permettent de simuler le pattern matching



# Références

- Site officiel : <https://golang.org/>
- The Go Programming Language, Alan A. A. Donovan et Brian W. Kernighan
- Tutoriel interactif : <https://tour.golang.org/>
- “REPL” en ligne : <https://play.golang.org/>
- Histoire de Go :  
[https://talks.golang.org/2012/splash.article#TOC\\_3](https://talks.golang.org/2012/splash.article#TOC_3).
- The TIOBE Programming Community index :  
<https://www.tiobe.com/tiobe-index/>



MERCI POUR VOTRE  
ATTENTION.