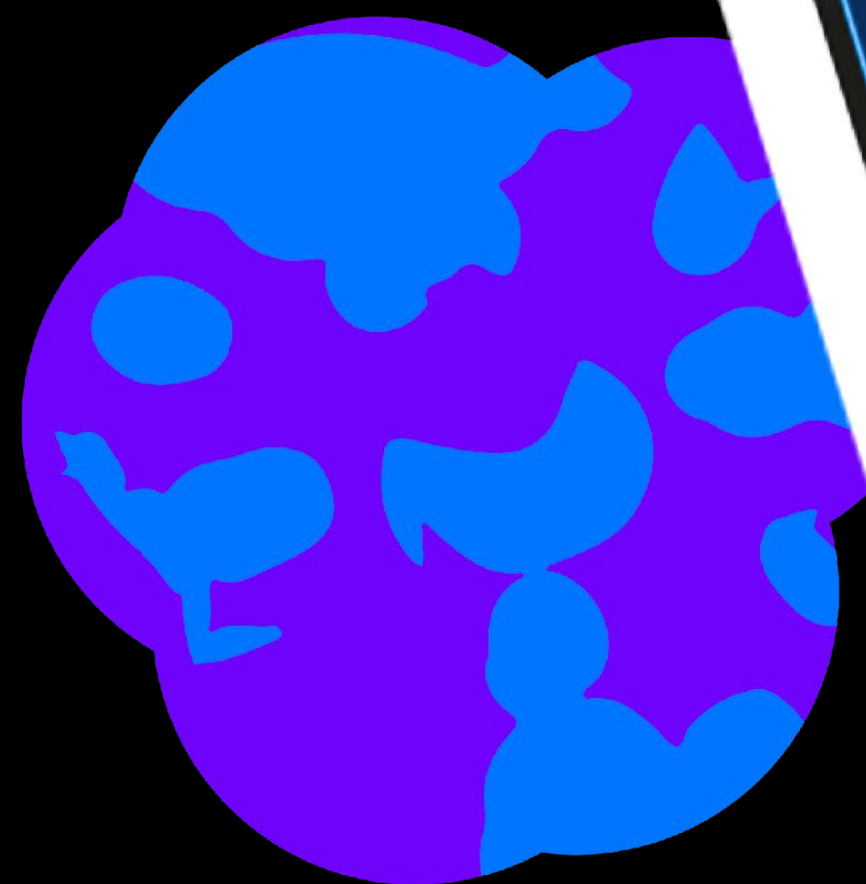




# Common Mistakes in Golang and How to Avoid Them





# Arrays and slices

**1**

**2**

**3**

**4**

**5**

**6**

**7**

**8**

# Arrays and slices

```
type slice struct {  
    array unsafe.Pointer  
    len    int  
    cap    int  
}
```

# Arrays and slices

- when creating a new slice, its `len == cap`, unless you specify a different value with the `make` function:  
    `slice1 := []int{1,2,3}`  
    `slice2 := make([]int, 3)`  
    и `slice1` и `slice2` will have `len == cap == 3`
- the rate of growth of slice capacity, starting from 1.20, has changed, and now smoothly decreases from x2 for slices with `cap < 512` to x1.3 for slices with `cap > 4096`

```
type slice struct {  
    array unsafe.Pointer  
  
    len    int  
  
    cap    int  
  
}
```

## Arrays and slices

```
func changeSliceValues(s []int) {  
    s[0] = 1  
}
```

```
func main() {  
    slice := []int{0}  
    fmt.Println(slice) // [0]  
    changeSliceValues(slice)  
    fmt.Println(slice) // [1]  
}
```

## Arrays and slices

```
func changeSliceValues(s []int) {  
    s[0] = 1  
    s = append(s, 2)  
    s[0] = 3  
}
```

```
func main() {  
    slice := []int{0}  
    fmt.Println(slice) // [0]  
    changeSliceValues(slice)  
    fmt.Println(slice) // [1]  
}
```

## Arrays and slices

```
func main() {  
    slice := []int{0, 0, 0}  
    newSlice := slice[0:2]  
    newSlice = append(newSlice, 1)  
    fmt.Println(slice) // [0, 0, 1]  
}
```

## Arrays and slices

```
func main() {  
    slice := []int{0, 0, 0}  
    newSlice := make([]int, 2)  
    copy(newSlice, slice)  
    newSlice = append(newSlice, 1)  
    fmt.Println(slice) // [0, 0, 0]  
    fmt.Println(newSlice) // [0, 0, 1]  
}
```



## Arrays and slices

```
func receiveArticle() string {  
    ...  
}
```

```
func consumeNewsArticles() {  
    for {  
        article := receiveArticle()  
        storeArticlePreview(getArticlePreview([]rune(article)))  
    }  
}
```

```
func getArticlePreview(article []rune) []rune {  
    return article[:100]  
}
```

## Arrays and slices

```
func receiveArticle() string {  
    ...  
}
```

```
func consumeNewsArticles() {  
    for {  
        article := receiveArticle()  
        storeArticlePreview(getArticlePreview([]rune(article)))  
    }  
}
```

```
func getArticlePreview(article []rune) []rune {  
    return article[:100]  
}
```



# Strings, runes and bytes

1

2

3

4

5

6

7

8

## Strings, runes and bytes

```
func main() {  
    hello := "Hello World"  
    helloRunes := []rune(hello)  
  
    fmt.Println(helloRunes[:5]) // [72 101 108 108 111]  
    fmt.Println(string(helloRunes[:5])) // Hello  
    fmt.Println(hello[:5]) // Hello  
}
```

## Strings, runes and bytes

```
func main() {  
    hello := "你好世界"  
    helloRunes := []rune(hello)  
  
    fmt.Println(helloRunes[:2])           // [20320 22909]  
    fmt.Println(string(helloRunes[:2])) // 你好  
    fmt.Println(hello[:2])               // ❖  
}
```

## Strings, runes and bytes

```
func main() {  
    hello := "你好世界"  
    fmt.Println(hello[:2]) // bytes  
    fmt.Println(len(hello)) // bytes  
  
    for i, c := range hello {  
        fmt.Println(i, c) // bytes index, rune  
    }  
}
```



# Channels

1

2

3

4

5

6

7

8

# Channels

Operation	Channel state	Result
Read	<code>nil</code>	Block
	Open and Not Empty	Value
	Open and Empty	Block
	Closed	<default value>, false
	Write Only	Compilation Error
Write	<code>nil</code>	Block
	Open and Full	Block
	Open and Not Full	Write Value
	Closed	<b>panic</b>
	Receive Only	Compilation Error
close	<code>nil</code>	<b>panic</b>
	Open and Not Empty	Closes Channel; reads succeed until channel is drained, then reads produce default value
	Open and Empty	Closes Channel; reads produces default value
	Closed	<b>panic</b>
	Receive Only	Compilation Error



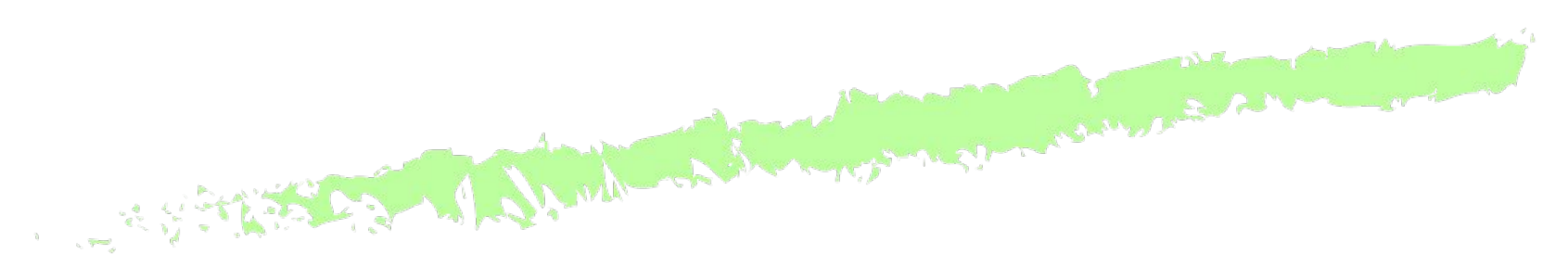
# Channels



Operation	Channel state	Result
Read		
	Closed	<default value>, false
Write		
	Closed	panic
close		
	Closed	panic

# Channels

“A sender can `close` a channel to indicate that no more values will be sent.”



# Channels

```
func writeToChan(ch chan<- int) {  
    ch <- 1  
    ch <- 2  
    ch <- 3  
    close(ch)  
}
```

```
func main() {  
    ch := make(chan int)  
  
    go writeToChan(ch)  
  
    for value := range ch {  
        fmt.Println(value)  
        // some logic  
    }  
}
```

# Channels

## func After

```
func After(d Duration) <-chan Time
```

After waits for the duration to elapse and then sends the current time on the returned channel. It is equivalent to `NewTimer(d).C`. The underlying `Timer` is not recovered by the garbage collector until the timer fires. If efficiency is a concern, use `NewTimer` instead and call `Timer.Stop` if the timer is no longer needed.

# Channels

```
func consumer(ch <-chan Event) {
    for {
        select {
            case event := <-ch:
                handle(event)
            case <-time.After(time.Minute * 15):
                fmt.Println("warning: no messages received")
        }
    }
}
```

# Channels

$$1\_000\_000 \text{ events} * \approx 200 \text{ bytes} = \approx 200 \text{ megabytes}$$

# Channels

“The underlying Timer is not recovered by the garbage collector until the timer fires. If efficiency is a concern, use `NewTimer` instead and call `Timer.Stop` if the timer is no longer needed”





# Goroutines

1

2

3

4

5

6

7

8



# Goroutines

```
func main() {
    digits := []int64{1, 2, 3, 4, 5}
    var sum int64 = 0
    var wg sync.WaitGroup
    for _, value := range digits {
        go func() {
            wg.Add(1)
            defer wg.Done()
            atomic.AddInt64(&sum, value)
        }()
    }

    wg.Wait()

    fmt.Println(sum)
}
```

# Goroutines

## Go Wiki: LoopvarExperiment

### Table of Contents

[How do I try the change?](#)

[What is the problem this solves?](#)

[What is the proposed solution?](#)

[Can this change break programs?](#)

[How often does the change break real programs?](#)

[Will the change make programs slower by causing more allocations?](#)

[If the proposal is accepted, how will the change be deployed?](#)

[Can I see a list of places in my code affected by the change?](#)

[My test fails with the change. How can I debug it?](#)

[Does this mean I don't have to write `x := x` in my loops anymore?](#)

[How can I send feedback?](#)

For Go 1.22, the Go team is considering changing the semantics of for loop variables to prevent unintended sharing in per-iteration closures and goroutines. Go 1.21 contains a preliminary implementation of the change, enabled by setting `GOEXPERIMENT=loopvar` when building your program. We invite anyone who wants to help us understand the effects of the change to try using `GOEXPERIMENT=loopvar` and let us know about any problems or successes encountered.

This page answers frequently asked questions about the change.



# Sync & atomic packages

1

2

3

4

5

6

7

8

## Sync & atomic packages

```
type WaitGroup struct {  
    noCopy noCopy  
  
    state atomic.Unit64  
    sema unit32  
}
```

## Sync & atomic packages

- 1) `.Add(delta int)` increases the semaphore value by the passed value
- 2) `.Done()` decreases the semaphore value by one
- 3) `.Wait()` blocks execution until the semaphore value becomes zero

```
type WaitGroup struct {  
    noCopy noCopy  
  
    state atomic.Unit64  
    sema  unit32  
}
```

## Sync & atomic packages

```
func main() {  
    digits := []int64{1, 2, 3, 4, 5}  
    var sum int64 = 0  
    var wg sync.WaitGroup  
    for _, value := range digits {  
        go func() {  
            wg.Add(1)  
            defer wg.Done()  
            atomic.AddInt64(&sum, value)  
        }()  
    }  
  
    wg.Wait()  
  
    fmt.Println(sum)  
}
```

## Sync & atomic packages

```
type Counter struct {
    m      sync.Mutex
    counters map[string]int
}

func (c Counter) increment(key string) {
    c.m.Lock()
    defer c.m.Unlock()
    c.counters[key]++
}

func (c Counter) IncrementMultiple(key string, n int) {
    for i := 0; i < n; i++ {
        c.increment(key)
    }
}
```

```
func main() {
    c := Counter{counters:
map[string]int{"key1": 0, "key2": 0}}

    go c.IncrementMultiple("key1", 100000)
    go c.IncrementMultiple("key1", 100000)

    time.Sleep(300 * time.Millisecond)
    fmt.Println(c.counters)
}
```

## Sync & atomic packages

```
type Counter struct {
    m      sync.Mutex
    counters map[string]int
}

func (c Counter) increment(key string) {
    c.m.Lock()
    defer c.m.Unlock()
    c.counters[key]++
}

func (c Counter) IncrementMultiple(key string, n int) {
    for i := 0; i < n; i++ {
        c.increment(key)
    }
}
```

```
func main() {
    c := Counter{counters:
map[string]int{"key1": 0, "key2": 0}}

    go c.IncrementMultiple("key1", 100000)
    go c.IncrementMultiple("key1", 100000)

    time.Sleep(300 * time.Millisecond)
    fmt.Println(c.counters)
}
```

**fatal error: concurrent map writes**  
**<goroutines stack>**  
**Process finished with the exit code 2**



# Sync & atomic packages

```
main.go x
1  package main
2
3  import (
4      "fmt"
5      "sync/atomic"
6  )
7
8  func main() {
9      var num int64 = 1
10     go func() {
11         for {
12             atomic.AddInt64(&num, delta: 1)
13         }
14     }()
15
16     for {
17         if atomic.LoadInt64(&num)%2 == 0 {
18             fmt.Println(atomic.LoadInt64(&num))
19             return
20         }
21     }
22 }
23
```

Run: go build test1 x

```
GOROOT=/Users/dskorolev/go/go1.21.4 #gosetup
GOPATH=/Users/dskorolev/go #gosetup
/Users/dskorolev/go/go1.21.4/bin/go build -o /private/var/folders/qh/w5_sb2356xn_q524l9hh5ll80000gn/T/GoLand/___go_build_test1 test1 #gosetup
/private/var/folders/qh/w5_sb2356xn_q524l9hh5ll80000gn/T/GoLand/___go_build_test1
287
Process finished with the exit code 0
```



# defer

1

2

3

4

5

6

7

8

# defer

```
type ProfileType string
const (
    SimpleProfile    ProfileType = "simple"
    InvestmentProfile ProfileType = "investment"
    BusinessProfile  ProfileType = "business"
)
type Profile struct {
    Type ProfileType
}
```

```
func (p *Profile) GetBalance() (balance
int) {
    switch p.Type {
    case BusinessProfile:
        return
        p.getBusinessProfileBalance()
    case InvestmentProfile:
        return
        p.getInvestmentProfileBalance()
    case SimpleProfile:
        return p.getSimpleProfileBalance()
    default:
        panic("unknown profile type")
    }
}
```

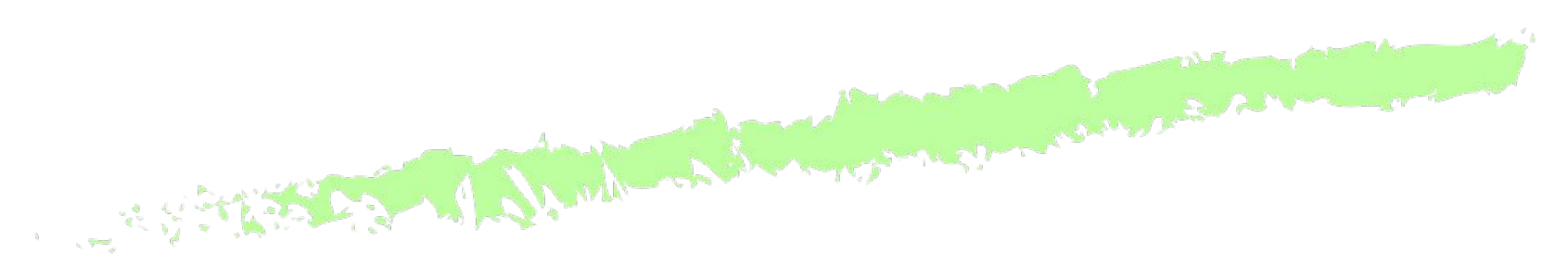
# defer

```
type ProfileType string
const (
    SimpleProfile    ProfileType = "simple"
    InvestmentProfile ProfileType = "investment"
    BusinessProfile  ProfileType = "business"
)
type Profile struct {
    Type ProfileType
}
```

```
func (p *Profile) GetBalance() (balance int) {
    defer fmt.Println("profile balance:", balance)
    switch p.Type {
    case BusinessProfile:
        return p.getBusinessProfileBalance()
    case InvestmentProfile:
        return p.getInvestmentProfileBalance()
    case SimpleProfile:
        return p.getSimpleProfileBalance()
    default:
        panic("unknown profile type")
    }
}
```

# defer

“The arguments to the deferred function (which include the receiver if the function is a method) are evaluated when the *defer* executes, not when the *call* executes”.



# defer

```
type ProfileType string
const (
    SimpleProfile    ProfileType = "simple"
    InvestmentProfile ProfileType = "investment"
    BusinessProfile  ProfileType = "business"
)
type Profile struct {
    Type ProfileType
}
```

```
func (p *Profile) GetBalance() (balance int) {
    defer func() {
        fmt.Println("profile balance:", balance)
    }()
    switch p.Type {
    case BusinessProfile:
        return p.getBusinessProfileBalance()
    case InvestmentProfile:
        return p.getInvestmentProfileBalance()
    case SimpleProfile:
        return p.getSimpleProfileBalance()
    default:
        panic("unknown profile type")
    }
}
```



# Interfaces

1

2

3

4

5

6

7

8

# Interfaces

```
type Requester interface {
    MakeRequest() int
}
type ConcreteRequester struct {
    someField int
}
func (r *ConcreteRequester) MakeRequest() int {
    return r.someField
}
```

```
func makeRequester(someVal int) Requester {
    var requester *ConcreteRequester
    if someVal > 0 {
        requester = &ConcreteRequester{someField: someVal}
    }
    return requester
}

func main() {
    requester := makeRequester(0)
    fmt.Println("got requester: ", requester)
    if requester == nil {
        fmt.Println("requester is nil")
    } else {
        fmt.Println("requester is not nil")
    }
}
```



# Interfaces

```
type Requester interface {
    MakeRequest() int
}
type ConcreteRequester struct {
    someField int
}
func (r *ConcreteRequester) MakeRequest() int {
    return r.someField
}
```

```
func makeRequester(someVal int) Requester {
    var requester *ConcreteRequester
    if someVal > 0 {
        requester = &ConcreteRequester{someField: someVal}
    }
    return requester
}

func main() {
    requester := makeRequester(0)
    fmt.Println("got requester: ", requester)
    if requester == nil {
        fmt.Println("requester is nil")
    } else {
        fmt.Println("requester is not nil")
    }
}
```

got requester: <nil>  
requester is not nil

# Interfaces

```
type eface struct {
    _type *_type
    data unsafe.Pointer
}

type iface struct {
    tab *itab
    data unsafe.Pointer
}

type itab struct {
    _type *_type
    ...
}
```

# Interfaces

```
type Requester interface {
    MakeRequest() int
}

type ConcreteRequester struct {
    someField int
}

func (r *ConcreteRequester) MakeRequest()
    int {
    return r.someField
}
```

```
func makeRequester(someVal int) Requester {
    var requester *ConcreteRequester
    if someVal > 0 {
        requester = &ConcreteRequester{someField: someVal}
    }
    return requester
}

func main() {
    requester := makeRequester(0)
    fmt.Println("got requester: ", requester)
    fmt.Printf("requester=(%T,%v)\n", requester, requester)
    if requester == nil {
        fmt.Println("requester is nil")
    } else {
        fmt.Println("requester is not nil")
    }
}
```

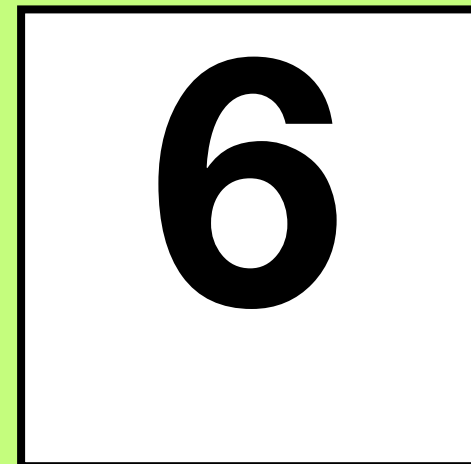
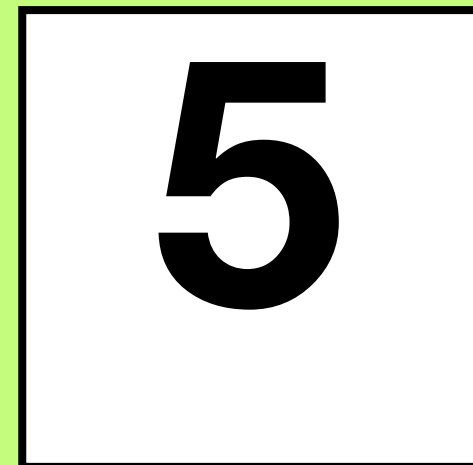
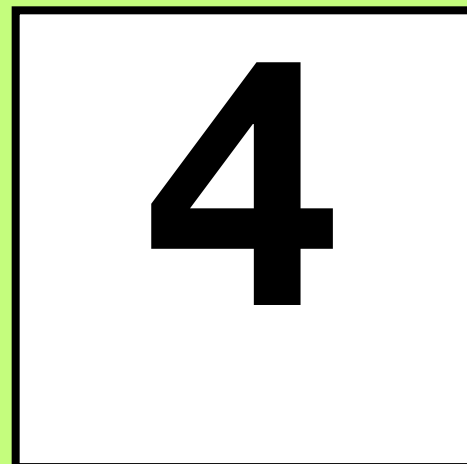
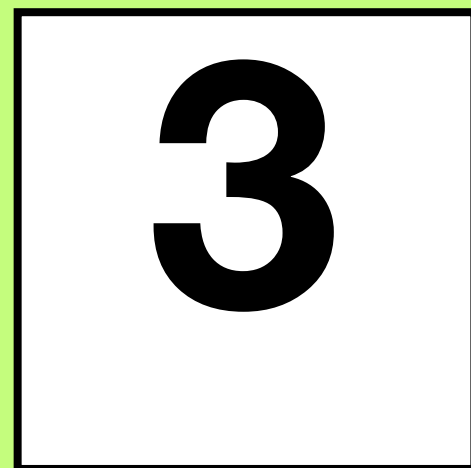
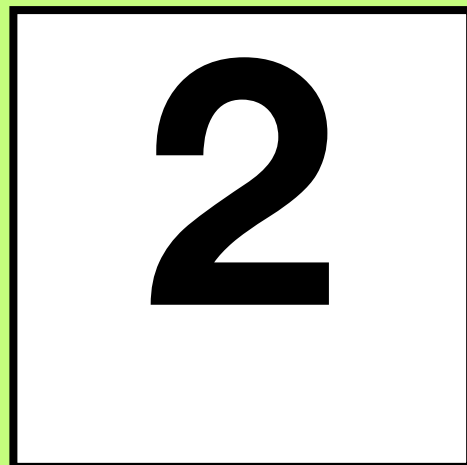
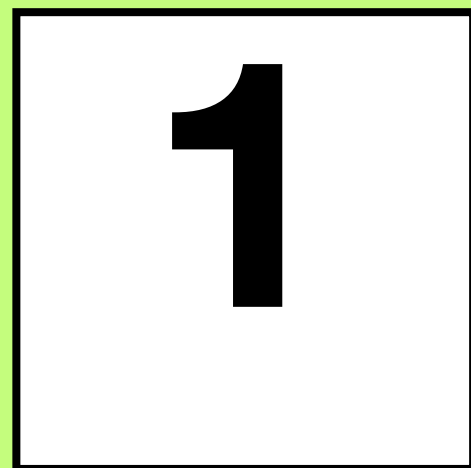
got requester: <nil>

requester=(\*main.ConcreteRequester,<nil>)

requester is not nil



# Vendoring peculiarities



# Vendoring peculiarities

```

  ▾ lib_example ~/Documents/avito/lib_example
    ▾ internal
      ▾ clients
        ▾ service1
          🐼 methods.go
          🐼 service1.go
        ▾ generated
          ▾ clients
            ▾ service1
              🐼 service1_client.go
        ▾ models
          🐼 processed_data.go
      📄 .gitignore
    > 📄 go.mod
    🐼 library.go
```

# Vendoring peculiarities

```

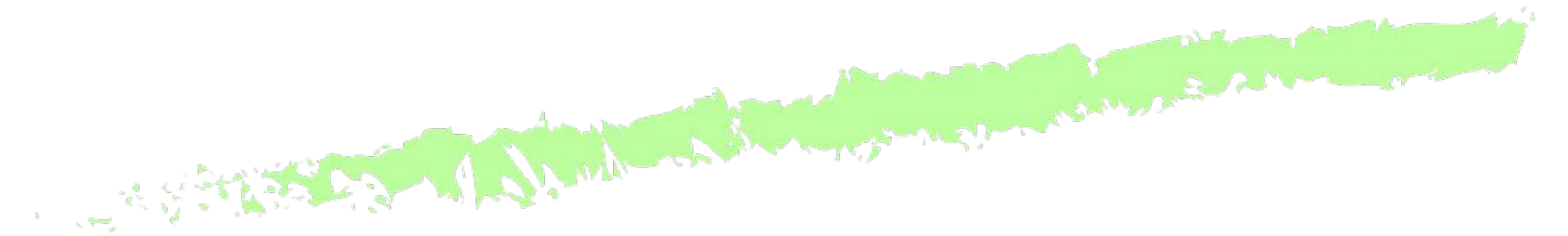
└─┬─ folder lib_example ~/Documents/avito/lib_example
   │
   └─ folder internal
      │
      └─ folder clients
         │
         └─ folder service1
            │
            ├── methods.go
            └── service1.go
      │
      └─ folder generated
         │
         └─ folder clients
            │
            └─ folder service1
               │
               └── service1_client.go
      │
      └─ folder models
         │
         └── processed_data.go
      │
      ├── .gitignore
      └─ go.mod
      └─ library.go
```

```

└─┬─ folder project_example ~/Documents/avito/project_example
   │
   └─ folder vendor
      │
      └─ folder github.com
         │
         └─ folder Jimiliani
            │
            └─ folder lib-example
               │
               └─ folder internal
                  │
                  └─ folder models
                     │
                     └── processed_data.go
                     │
                     ├── .gitignore
                     └── library.go
                  │
                  └── modules.txt
      │
      └─ go.mod
      └─ main.go
```

## Vendoring peculiarities

“The go mod vendor command constructs a directory named vendor in the main module’s root directory containing copies of all packages needed to build and test packages in the main module”.





# Common Mistakes in Golang and How to Avoid Them

