

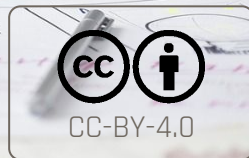
# Go言語 初心者向けハンズオン

Sep 14, 2018 / #techdo #golang #mediado #redish

written by @k\_h\_sissp

@yukpiz

proofread @ariaki4dev



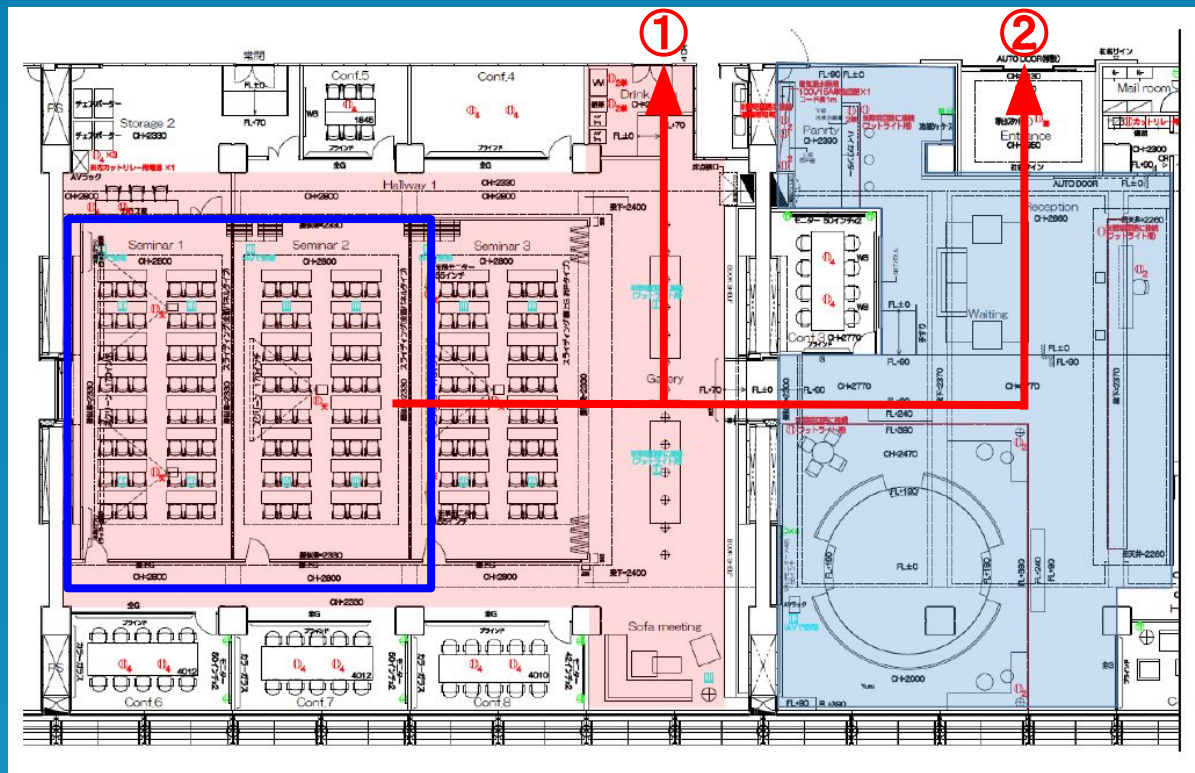
# Information

---

ご案内

# 避難経路のご案内

会場  
前方



- ①会場後方勝手口  
(ベンチスペース脇)
- ②中央エントランス

※非常時は係員による避難指示に従ってください

# 設備のご案内

---

## ゲスト用電源

足元にコンセントがございます

※数に限りがありますので譲り合ってください

## ゲスト用Wi-Fi

<b>ssid</b>	md-guest
<b>password</b>	12345678

# ご案内

---

## ご案内とお願い

- イベント内容は後日任意の媒体にて公開させて頂くことがあります
- イベントレポート作成のため、写真/動画を撮影いたします
- イベント中は自由にご飲食いただけます
- 体調不良等ございましたらスタッフまでお申し付けください

# Tech Doについて

## Tech Doは自主勉強会です

株式会社メディアドゥの技術者が自主的に始めた勉強会です！

毎月第2金曜に実施していますので興味ある方は @ariaki4dev までご連絡ください！

業務時間内に自由学習



学習



発表

# コミュニケーション

---



[bit.ly/techdo-slack](https://bit.ly/techdo-slack)

#support-201809



[#techdo](#)

# Today's **Timeline**

本日のスケジュール

19:30	オープニング & 乾杯
19:40	<b>Go</b> 言語の基本構文
20:20	問題説明
20:35	解答時間（自由時間）
21:25	クロージング



# Speakers



**kentfordev**

@k\_h\_sissp



**yukpiz**

@yukpiz



Hello, **Gophers!**

.....  
はじめに

# Hello, Gophers!

```
package main

import "fmt"

func main() {
    fmt.Println("Hello, Gophers!")
}
```

※以下のURLよりオンライン実行できます

<https://play.golang.org/p/NpV07jhxbG>

# 実行方法

---

## 即時実行

```
go run hello.go
```

## コンパイルと実行

```
go build hello.go
```

```
./hello
```

# Goについて

---

## Goとは

- 2009年に発表、2012年にリリース
- 静的型付け
- コンパイル言語

## Goの特徴

- 並列実行性
- ガベージコレクション (Mark And Sweep)
- 簡潔性 (言語構文・機能)
- 実行速度



# Language **Structure**

---

言語構造

# この時間で話す内容

✓ 変数と定数

✓ 制御構文

✓ 関数

✓ 配列

✓ 構造体

✓ インポート

✓ ポインタ

✓ パッケージ

✓ GOPATH



# Variables & Constants

変数と定数



# 変数

## 変数宣言

変数 "total" を int 型で宣言する例

```
// 1. 変数宣言  
var total int  
// 2. 変数宣言と初期値の代入  
var total int = 100  
// 3. 変数宣言と初期値の代入（型を省略）  
var total = 100  
// 4. 変数宣言と初期値の代入（varを省略）  
total := 100
```

# 変数

## 変数再宣言時のエラー

同一スコープ内で同名変数を再宣言する事ができない例

```
var total int = 100
```

変数の再宣言はコンパイルエラー

```
var total int = 200
```

```
total := 300
```

// NG: total redeclared in this block

// NG: no new variables on left side of :=

```
total, i := 400, 500 // OK: 再代入として扱われる
```

※未宣言の変数と列記した場合、自動的に再代入として扱われる

# 変数

## 変数再代入

変数 "total" に対して値を再代入する例

```
var total int = 100
```

```
total = 200
```

# 変数

## 変数型

代表的な変数型一覧

型	説明	型	説明
uint [ 8   16   32   64 ]	符号なし数値	byte	uint8
int [ 8   16   32   64 ]	符号あり数値	rune	int32
float [ 32   64 ]	IEEE-754 浮動小数値	bool	論理値
complex [ 64   128 ]	複素数	string	文字列
uint	uint32 又は uint64	error	エラー
int	int32 又は int64		

# 定数

## 定数定義

定数 "priority" を int 型で定義する例

```
// 1. 定数宣言  
const priority int = 1  
  
// 2. 定数宣言（型を省略）  
const priority = 1
```

# 変数と定数

---

## ブランク識別子

アンダースコア(\_)を使って記述されるブランク識別子は、他の識別子と同様に宣言で使用されますが、その宣言ではバインドは行われません。

## 未使用変数

コード内に未使用変数が存在する場合、コンパイルに失敗します  
(*name declared and not used*)

# ブランク識別子

## ブランク識別子 ①

関数からの戻り値を破棄する例

```
func main() {  
    v, _ := hoge()  
}  
func hoge() (string, string) {  
    return "kentfordev", "yukpiz"  
}
```

※関数の記述方法については後述します



# ブランク識別子

## ブランク識別子 ②

配列の添え字を破棄する例

```
items := [...]string{"kentfordev", "yukpiz"}

for _, item := range items {
    fmt.Println(item)
}
```

※配列の記述方法については後述します

# Control flow statement

制御構文



# ifステートメント

## 基本形 ①

else がない例

```
if x > max {  
    x = max  
}
```

括弧は不要

評価式  $x > \text{max}$  が成り立つ場合は  $x$  に対して  
 $\text{max}$  を代入する

else がある例

```
if x < max {  
    return x  
} else {  
    return max  
}
```

評価式  $x < \text{max}$  が成り立つ場合は  $x$  を返し、  
成り立たない場合は  $\text{max}$  を返す

# ifステートメント

## 基本形 ②

else if を使用する例

```
if x > 10 {  
    return 10  
} else if x > 5 {  
    return 5  
} else {  
    return 0  
}
```

評価式  $x > 10$  が成り立つ場合は 10 を返す

評価式  $x > 5$  が成り立つ場合は 5 を返し、それ以外の場合は 0 を返す

# ifステートメント

## if 構文内での変数宣言

if 構文内で変数を宣言する例

```
if x := f(); x > 10 {  
    return x  
} else {  
    return 10  
}
```

xを宣言/代入し、ブロック内で使用

まずはじめに x を宣言し、関数 f() の戻り値を代入する

評価式  $x > 10$  が成り立つ場合は x を返し、それ以外の場合は 10 を返す

# switchステートメント

## 基本形

switch 式が存在する例

```
switch x {  
  case 0:  
    f1()  
  case 1, 2, 3: 自動で break  
    f2()  
  default:  
    f3()  
}
```

switch 式が省略された例

```
switch {  
  case x < 10:  
    f1()  
  case x > 10:  
    f2()  
  default:  
    f3()  
}
```


条件に合致する  
処理を実行

# switchステートメント

## フォールスルー

フォールスルーの例

```
switch x {  
  case 0:  
    f1()  
    fallthrough  
  case 1, 2, 3:  
    f2()  
    fallthrough  
  default:  
    f3()  
}
```



次ケースへ処理を継続

# forステートメント

## 基本形

典型的な for ステートメント

```
for i := 0; i < 10; i++ {  
    sum += i  
}
```

変数 `i` を 1 ずつ増やしながら 0..9 の間ループ

無限ループ for ステートメント

```
for ;; {  
    sum++  
    if sum > 10 {  
        break  
    }  
}
```

無限ループ内で変数 `sum` を操作

評価式 `sum > 10` が成り立つ場合に `break` する



# forステートメント

## 発展形

評価式 for ステートメント

```
for (sum < 10) {  
    sum++;  
    fmt.Println("sum is ", sum)  
}
```

while 文が存在せず for で同様の処理が行える

評価式 `sum < 10` が成り立つ間ループ

※評価式自体を省略した場合は無限ループ構造になる

# Function

関数



# 関数

## 関数の定義 ①

典型的な関数の定義例

```
func area(w int, h int) int {  
    return w * h  
}
```

戻り値型を指定

引数と型を指定

return文で戻り値を返却

関数 `area(w, h)` を定義し、コールされた際に `w * h` の計算結果を返却する

# 関数

## 関数の定義 ②

関数によって複数值が返却される例

```
func main() {  
    a, b := calc(10, 20)  
    fmt.Println("total", a, "average", b)  
}  
  
func calc(x int, y int) (int, int) {  
    return x + y, (x + y) / 2  
}
```

複数の戻り値を一括で変数宣言/代入

戻り値を複数指定

return文で戻り値を複数返却

関数 `calc(x, y)` を定義し、コールされた際に合計 `(x+y)` 及び平均 `(x+y/2)` を返却する

# 関数

## 関数の定義 ③

関数を変数に代入する例

```
// 関数の定義  
myfunc := func() string { return "x" }  
// 関数の実行  
myfunc()
```

関数は JavaScript 等と同様に、変数に代入して利用できる

## 関数の定義 ④

戻り値パラメータに名前を設定する例

```
func main() {  
    a, b := calc(10, 20)  
    fmt.Println("total", a, "average", b)  
}
```

```
func calc(x int, y int) (sum int, avg int) {  
    sum = x + y  
    avg = (x + y) / 2  
    return  
}
```

名前を設定して使用

代入によって戻り値を設定

関数の定義 ② と同様の動作を行う関数 `calc(x, y)` が定義される

# Array, Slice & Map

配列 / スライス / マップ



# 配列

## 配列定義

配列を []string 型で定義する例

```
// 1.配列宣言  
var v1 = [10]string  
// 2.配列宣言  
v2 := [10]string{}  
// 3.配列宣言（初期化あり）  
v3 := [3]string{"Sun", "Mon", "Tue"}  
// 4.配列宣言（初期化あり/自動サイズ）  
v4 := [...]string{"Sun", "Mon", "Tue"}
```

初期データを設定

要素数を省略



# 配列

## 配列データ取得

配列データを取得/表示する例

```
s := [...]string{"Sun", "Mon", "Tue"}
```

```
fmt.Println(s[0]) // Sun
```

```
fmt.Println(s[3]) // ERROR: out of bounds
```

```
fmt.Println(len(s)) // 3
```

添字は 0 から始まる

範囲外の添字へのアクセスはエラー

len() メソッドによって要素数を取得

# スライス

## スライス定義 ①

配列とスライスの定義例

// 1. 配列宣言      要素数を**設定** → 配列：固定長  
a := [3]string{"a", "b", "c"}

// 2. スライス宣言      要素数**未設定** → スライス：可変長  
s := []string{"a", "b", "c"}

# スライス

## スライス定義 ②

配列とスライスの定義例（値の省略時）

// 1. 配列宣言

`a := [3]string{}`

省略時は**初期値**が設定される

(string : "", int : 0, bool : false)

// 2. スライス宣言

`s := []string{}`

省略時は**空集合**となる

# スライス

## 要素の取り出し

配列からスライスを取り出す例

```
a := [5]string{"a", "b", "c", "d", "e"}
```

```
s1 := a[1:2] // {b, c}
```

[開始index : 終了index]

```
s2 := a[1:] // {b, c, d, e}
```

```
s3 := a[:2] // {a, b}
```

```
s4 := a[:] // {a, b, c, d, e}
```

省略時はそれぞれ先頭/末尾となる

# スライス

## 要素の上書き

スライスに対するデータ上書きの例

```
a := [3]string{"a", "b", "c"}
```

```
s := a[:]
```

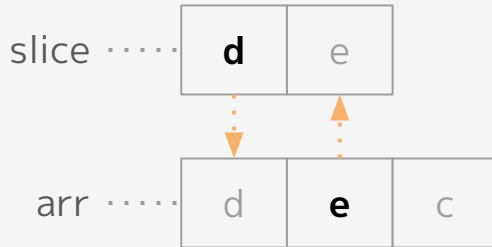
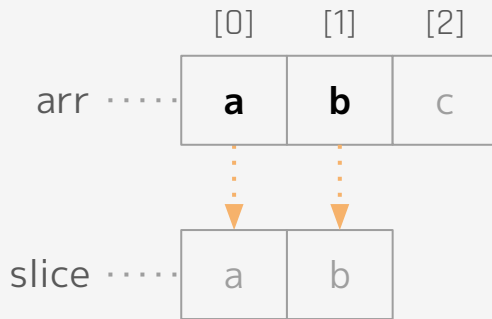
```
s[0] = "d"
```

```
a[1] = "e"
```

```
fmt.Println(a) // {d, e, c}
```

```
fmt.Println(s) // {d, e, c}
```

スライスは元の配列を参照する為、**互いの変更が反映される**



# スライス

## 要素の追加

スライスに要素を追加する例

```
a := []string{"a", "b", "c"}  
  
a = append(a, "d")  
  
fmt.Println(a) // {a, b, c, d}
```

スライスへの要素追加は `append` メソッドで行う

# スライス

## キャパシティ ①

コピーしたスライスの値を書き換える例

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

```
b[0] = "x"
```

```
fmt.Println(a) // {x, b, c}  
fmt.Println(b) // {x, b, c}
```

左記と比べて `append()` を追加した例

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

```
a = append(a, "d")  
b[0] = "x"
```

```
fmt.Println(a) // {a, b, c, d}  
fmt.Println(b) // {x, b, c}
```

スライスには「**キャパシティ**（保持できる上限個数）」という概念があり、これを超過した場合は元データとは**切り離され**、別のスライスとして生成される

# スライス

## キャパシティ ②

キャパシティを確認する例

```
a := []string{"a", "b", "c"}  
fmt.Println(len(a)) // 3  
fmt.Println(cap(a)) // 3
```

キャパシティは初期サイズ 3 となる

```
a = append(a, "d")  
fmt.Println(len(a)) // 4  
fmt.Println(cap(a)) // 6
```

キャパシティは倍の 6 に拡張される  
(※別スライスが生成される)

- スライスの長さを取得する → `len()`
- スライスのキャパシティを取得する → `cap()`




# マップ

## マップ定義

典型的なマップの定義例

```
ages := map[string]int {  
    "bob": 25,  
    "alice": 24,  
}
```



必須

マップへのアクセス例

```
// 取得  
age := ages["alice"]  
// 上書  
ages["bob"] = 27  
// 追加  
ages["yukpiz"] = 28  
// 削除  
delete(ages, "alice")
```

# Type & Structure

型と構造体



# 型

## 型定義

典型的な型の定義例

```
type hoge int  
type foobar string
```

```
var a hoge = 123  
var b foobar = "abc"
```

独自の型名で型を定義し利用可能

# 構造体

## 構造体定義 ①

典型的な構造体の定義例

```
type user struct {  
    Name string  
    Age int  
}
```

構造体を用いて変数を宣言する例

```
// データを指定せず初期化  
yukpiz1 := user{}  
// 一部のデータを指定して初期化  
yukpiz2 := user{  
    Name: "yukpiz",  
}  
// 全てのデータを指定して初期化  
yukpiz3 := user{  
    Name: "yukpiz",  
    Age: 28,  
}
```

必須

# 構造体

## 構造体定義 ②

構造体定義と初期化を同時に行う例

```
yukpiz := struct {  
    Name string  
    Age  int  
}{  
    Name: "yukpiz",  
    Age: 28,  
}
```

構造体のフィールド値を操作する例

```
// 構造体のフィールド値を更新  
yupiz.Name = "yukpiz"  
  
// 構造体のフィールド値を取得  
name := yukpiz.Name
```

# Import

インポート



# インポート

## インポート宣言

パッケージ "fmt" 及び "strings" のインポート例

```
// インポートを1件ずつ宣言
import "fmt"
import "strings"

// インポートをまとめて宣言
import (
    "fmt"
    "strings"
)
```

# インポート

## インポート宣言

パッケージ名を別名定義する例

```
import f "fmt"
func main(){
    f.Println("Hello, gophers!")
}
```

別名定義して利用

パッケージ名を省略する例

```
import . "fmt"
func main(){
    Println("Hello, gophers!")
}
```

パッケージ名を省略



# Pointer

ポインタ



# ポインタ

## ポインタの基本

典型的なポインタ宣言の例

```
var p *int
var n int = 10
```

ポインタ型の変数

```
p = &n
```

変数 n のアドレスを p に格納

```
fmt.Println(*p)
```

間接参照により n の値を取得、結果は **10**

# ポインタ

## デリファレンス ①

関数へ int ポインタを渡す例

```
func main() {  
    a := 1  
    b := 1  
    add1(a)  
    add2(&b)  
    fmt.Println(a, b) // {1, 2}  
}
```

a: 値渡し

b: ポインタ渡し

結果は { 1, 2 }

```
func add1(x int) { x = x + 1 }  
func add2(y *int) { *y = *y + 1 }
```

x: コピーされた変数を操作

y: 参照元の b を操作

# ポインタ

## デリファレンス ②

関数へ struct ポインタを渡す例

```
type user struct {  
    Name string  
}  
  
func main() {  
    a := user{ Name: "kent" }  
    b := user{ Name: "yukpiz" }  
    rename1(a)  
    rename2(&b)  
    fmt.Println(a.Name, b.Name) // {kent, ariaki}  
}  
  
func rename1(x user) { x.Name = "ariaki" }  
func rename2(y *user) { y.Name = "ariaki" }
```

結果は { kent, ariaki }

x: コピーされた変数を操作  
y: 参照元の b を操作

# Package

パッケージ



# パッケージ

## パッケージとは

- Go 言語が参照可能な、名前付けされたプログラムの名前空間
- パッケージが別のプログラムは import しないと使えない
- プロジェクトの中で使用するプログラムは、自分で名前をつけてパッケージを分ける
  - 作成したパッケージは相対パスで参照する

### プログラム配置

file : main.go

dir : mypackage

file : mypackage.go



mypackage.go

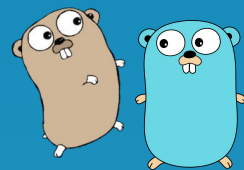
package mypackage を記述

main.go

import "../mypackage" を記述

# GOPATH

パス設定



# GOPATH

## GOPATHとは

- import 文の解決に使われるパスが格納される環境変数
- 外部パッケージのダウンロード先としても使われる
- go/build パッケージで実装されている
- 基本的には以下の 2 点を設定するだけでよい

```
export GOPATH=$HOME/go
```

```
export PATH=$PATH:$GOPATH/bin
```



# Solving **Exercises**

練習問題

# 解答時間について



# テーマ

---

## Web APIサーバを作る

Go言語は様々な用途で活用する事ができます。

本日は皆さんがより親しみやすく、そして楽しく学べる入り口としてこのテーマを選択しました。

## Web APIとは

[ API : Application Programming Interface ]

Webブラウザ等からのリクエストに応じてJSON等のデータを返却します。

本日は `http://localhost:8080` でアクセスできるように開発を進めていきます。

# Web APIサーバの準備

---

## Web APIサーバの作り方

以下の記事を参考にして、初めてのWeb APIサーバを実行してください。

<https://qiita.com/vukpiz/items/64e2874fd37a9dc7365d>

# 本日の目標

1. Go言語の構文に慣れる
2. Web APIサーバ実装方法を知る

# 練習問題

---



#1: FizzBuzz API



#2: プロフィール取得API



#3: プロフィール保存API



#4: プロフィール取得API拡張



#5: APIクライアント



#6: HTMLを返却

# 練習問題 #1

---

## FizzBuzz APIを作ろう

- GET `http://localhost:8080/FizzBuzz/:num`
- 1 から :num で指定された数値を元に、以下を出力してみましょう
  - 3 で割り切れる場合 → Fizz
  - 5 で割り切れる場合 → Buzz
  - 3 及び 5 で割り切れる場合 → FizzBuzz!
  - 上記以外の場合 → 数値
- :num が正の整数でない場合は、HTTP 1.1/400 を返却しましょう

# 練習問題 #2

## プロフィール取得APIを作ろう

- GET `http://localhost:8080/Profile/:name`
- `:name` が Bob と Alice ならプロフィールを JSON 形式で返しましょう
- `:name` が上記以外の場合は、HTTP 1.1/400 を返却しましょう

### Bob

Name: Bob

Age: 25

Gender: Man

Favorite Foods: Hamburger, Cookie, Chocolate

### Alice

Name: Alice

Age: 24

Gender: Woman

Favorite Foods: Apple, Orange, Melon



# 練習問題 #3

## プロフィール保存APIを作ろう

- POST `http://localhost:8080/Profile`
- 今回はデータベースを使わず、変数にプロフィールを保持しましょう
- `:name` が同一のユーザーは登録させず、HTTP 1.1/400 を返却しましょう
- パラメータに不備がある時も同様に登録させず、HTTP 1.1/400 を返却しましょう
- プロフィール格納に成功した場合は、HTTP 1.1/201 を返却しましょう
- Body パラメータとして以下を含むリクエストを受け付けましょう

<code>name</code>	<code>string</code>
<code>age</code>	<code>int</code>
<code>gender</code>	<code>string</code>
<code>favorite_foods</code>	<code>[]string</code>

# 練習問題 #4

---

## プロフィール取得APIを拡張しよう

- 練習問題 #2 の API を元に機能を拡張しましょう
- 練習問題 #3 の登録データを返却できるようにしましょう

# 練習問題 #5

## APIクライアントを作ろう

- 練習問題 #3 + #4 の API を呼び出せるクライアントを作しましょう
- CLI (コマンドライン上) で動作するクライアントにしましょう
- コマンドライン引数を扱うために flag パッケージを使いましょう
- 以下のように呼び出せるようにしてみましょう

### プロフィールを保存

```
go run main.go -name yukpiz -age 28 -gender Man -favorite-foods "Apple Orange"
```

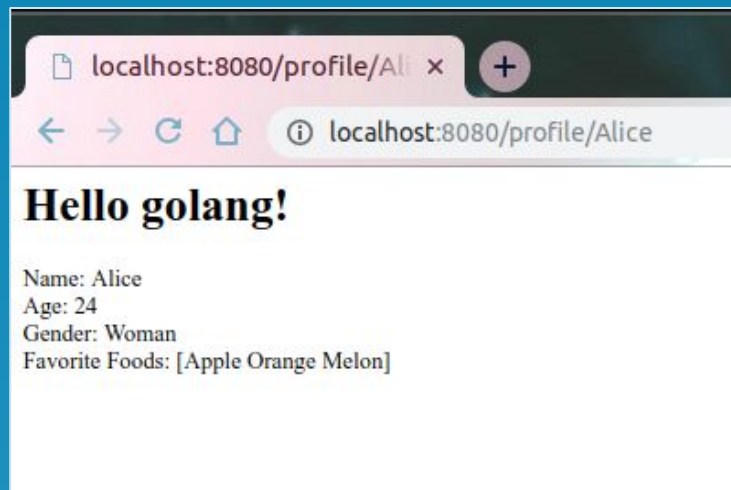
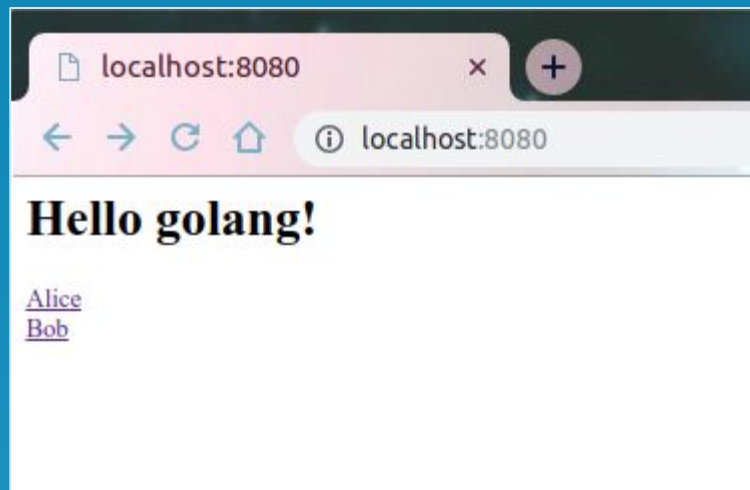
### プロフィールを取得

```
go run main.go -name yukpiz
```

# 練習問題 #6

## HTMLを返そう

- 練習問題 #3 + #4 を使い、以下の Web ページを返すサーバプログラムを書いてみましょう
- html/template パッケージを使いましょう



解答時間      -21:25 まで

※質問は以下のいずれかの方法でご連絡ください

① お近くの腕章をつけたスタッフ

② Slack ( [bit.ly/techdo-slack](https://bit.ly/techdo-slack) ) #support-201809 チャンネル

# Conclude & Closing

---

さいごに

※解答が終わるまで以降のページを**見ないで**ように注意してね！

# 解答について

---

解答例は以下URLよりご覧ください

<https://qiita.com/yukpiz/items/384093ce65c56451bd97>

## ハンズオン後のサポートについて

閉会後に引き続きご質問がある場合、Slackでサポートします

以下のワークスペースにジョインし、**#support-201809**チャンネルでお気軽にご質問ください

[bit.ly/techdo-slack](https://bit.ly/techdo-slack)

# redish

リピーターに長く愛される  
そんなお店づくりを支援したい

- \* **Marketing** Support
- \* **Concierge** Support
- \* **Opening** Support

for Restaurant !!

yukpiz  
@yukpiz





We deliver more content for everybody to enjoy

ひとつでも多くのコンテンツを  
ひとりでも多くの人へ



kentfordev

@k\_h\_sissp



# スタッフに拍手を！



kentfordev

@k\_h\_sissp



Manami Kayamori

@kaya\_salon



yukpiz

@yukpiz



ariaki

@ariaki4dev

Yohei Takeda

Haruka Okuno

Taiki Mimori

Nami Ikeda

Kumiko Hakamada

& You !!!

アンケートにご協力ください



<http://bit.ly/go20180914>

# Appendix

---

付録

# 参考資料

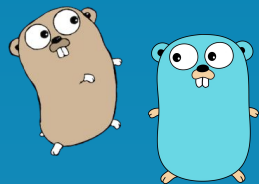
---

## Go初心者向け資料（英語）

- [How to Write Go Code](https://golang.org/doc/howto/writegocode) - golang.org
- [Effective Go](https://golang.org/doc/effective_go) - golang.org
- [Whispering Gophers](https://www.dmitrybaranovskiy.github.io/whispering-gophers/)
- [Go Wiki](https://github.com/golang/go/wiki) - github.com

## Go初心者向け資料（日本語）

- [A Tour of Go](https://golang.org/doc/tour)
- [Goプログラミング言語仕様](https://golang.jp) - golang.jp
- [Goプログラミング言語のチュートリアル](https://golang.jp/tutorial) - golang.jp
- [はじめてのGo](https://gihyo.jp) - gihyo.jp
- [Go Web プログラミング](#)



The Go gopher was designed by Renée French.