



Go picine

Go 00

*Summary:* このドキュメントは *Go picine @ 42Tpkyo* の *Go 00* のモジュール用の課題です。

# Contents

<b>I</b>	<b>Instructions</b>	<b>2</b>
<b>II</b>	<b>Foreword</b>	<b>3</b>
<b>III</b>	<b>Exercise 00 : Hello World</b>	<b>4</b>
<b>IV</b>	<b>Exercise 01 : Variables</b>	<b>5</b>
<b>V</b>	<b>Exercise 02 : isEmailAddress</b>	<b>6</b>
<b>VI</b>	<b>Exercise 03 : Create stairs</b>	<b>7</b>
<b>VII</b>	<b>Exercise 04 : Create function</b>	<b>8</b>
<b>VIII</b>	<b>Exercise 05 : a slice of ...</b>	<b>10</b>
<b>IX</b>	<b>Exercise 06 : echo!!!</b>	<b>12</b>

# Chapter I

## Instructions

- このページのみを参考にしてください。噂を信用しないで下さい。
- この文章は、提出前に変更になる可能性があります。十分に注意して下さい。
- ファイルとディレクトリへの権限があることをあらかじめ確認して下さい。
- 課題は全て提出手順に従って行って下さい。
- 課題の確認と評価は、あなたのクラスメイトが行います。
- 課題はgofmtと呼ばれるGoの公式code formatterによっても確認・評価されます。

```
$ gofmt -d test.go
$
```

- 上記のコードを実行して、コード変更の出力がされた場合は、規約に則ってないとして、0点になるので、注意してください。
- コンパイルできなかった場合は、評価は0です。
- package名はすべてmainとしてください。
- built-in関数の使用は認められています。
- goのversionは1.16.3を使用してください。
- あなたを助けてくれるのはGoogle / 人間 / インターネット / ... と呼ばれているものです。
- GoはGoogleが開発した言語なので、Googleに聞きましょう。
- Gopherくんを調べると、ヒントが得られるかも。
- 評価は42のiMacかGuacamoleで行われます。



golang.org

# Chapter II

## Foreword

"Go is an open source programming language that makes it easy to build simple, reliable, and efficient software." (From the Go web site at [golang.org](http://golang.org))

Go was conceived in September 2007 by Robert Griesemer, Rob Pike, and Ken Thompson, all at Google, and was announced in November 2009.

The goals of the language and its accompanying tools were to be expressive, efficient in both compilation and execution, and effective in writing reliable and robust programs.

Good luck.


Thank you for your help.

@42Tokyo

khiroshi  
tayamamo  
syudai  
ryhosoka  
ymiyata  
dnakano  
diuchi

# Chapter III

## Exercise 00 : Hello World


	Exercise 00
Hello World	
提出するディレクトリ : <i>ex00/</i>	
提出するファイル : <i>hello-world.go</i>	
使用可能なパッケージ : <i>fmt</i>	

- "Hello World!"と標準出力するプログラムを書いてください.

```
$ go run hello-world.go | cat -e
Hello World!$
$
```

# Chapter IV

## Exercise 01 : Variables

	Exercise 01
What kind of variables are there?	
提出するディレクトリ : <i>ex01/</i>	
提出するファイル : <i>vars.go</i>	
使用可能なパッケージ : <i>fmt</i>	

- 以下のように標準出力するプログラムを書いてください。

```
$ go run vars.go | cat -e
42 is a string.$
42 is a uint.$
42 is an int.$
42 is a uint8.$
42 is an int16.$
42 is a uint32.$
42 is an int64.$
false is a bool.$
42 is a float32.$
42 is a float64.$
(42+0i) is a complex64.$
(42+21i) is a complex128.$
{} is a main.FortyTwo.$
[42] is a [1]int.$
map[42:42] is a map[string]int.$
0x0 is an *int.$
[] is a []int.$
<nil> is a chan int.$
<nil> is a <nil>.$
$
```

# Chapter V

## Exercise 02 : isEmailAddress

	Exercise 02
	isEmailAddress
	提出するディレクトリ : <i>ex02/</i>
	提出するファイル : <i>isEmailAddress.go</i>
	使用可能なパッケージ : <i>All</i>

- コマンドライン引数で1つ以上の文字列を受け取ってください。0の時は

```
Invalid argument
```

と改行付き出力で怒ってください。

- もしその文字列が有効なメールアドレスだったら

```
<argument> is a valid email address.
```

と改行付きで出力し(もちろん<argument>のところは置き換えてください)、そうでなければ

```
<argument> is NOT a valid email address.
```

とこれまた改行付きで出力してください。( <argument> は置き換えて！ )

- ただし、有効なメールアドレスとは以下の正規表現で表せて256文字以下のものとします。

```
"[\w\-\.\_]+\@[\w\-\.\_]+\.[A-Za-z]+"
```


- 例

```
$ go build isEmailAddress.go
$ ./isEmailAddress marvin@student.42tokyo.jp abc@def.123 | cat -e
marvin@student.42tokyo.jp is a valid email address.$
abc@def.123 is NOT a valid email address.$
```

```
$ ./isEmailAddress | cat -e
Invalid argument$
$
```

# Chapter VI

## Exercise 03 : Create stairs

	Exercise 03
Create stairs	
提出するディレクトリ : <i>ex03/</i>	
提出するファイル : <i>createStairs.go</i>	
使用可能なパッケージ : <i>fmt, os, strconv</i>	


- "\*"を使って例のような階段を標準出力するプログラムを作りなさい。
- 最大で"\*"を使える回数がコマンドライン引き数から渡されます。
- 有効なコマンドライン引き数が必ず1つ渡されます。引き数を検査する必要はありません。
- コマンドライン引き数で与えられる値は0以上10000以下の整数です。
- 例

```
$ go build createStairs.go
$ ./createStairs 0 | cat -e
$ ./createStairs 1 | cat -e
$ *$
$ ./createStairs 4 | cat -e
$ *$
$ **$
$ ./createStairs 10 | cat -e
$ *$
$ **$
$ ***$
$ ****$
$ ./createStairs 12 | cat -e
$ *$
$ **$
$ ***$
$ ****$
$
```



# Chapter VII

## Exercise 04 : Create function

	Exercise 04
Create function	
提出するディレクトリ : <i>ex04/</i>	
提出するファイル : <i>calculation.go</i>	
使用可能なパッケージ : <i>fmt, os, strconv</i>	

- 以下のmain関数と標準出力を満たすような関数 `calculationStr` を作成してください。

```
const ERROR_MSG string = "Arguments is invalid."

func main() {
    s, ok := calculationStr(os.Args)
    if !ok {
        fmt.Println(ERROR_MSG)
        os.Exit(1)
    }
    fmt.Print(s)
}
```

```
$ go build calculation.go
$ ./calculation 12 4 | cat -e
sum: 16$
difference: 8$
product: 48$
quotient: 3$
$
```


```
$ ./calculation a 4 | cat -e
Arguments is invalid.$
$
```



エラーが想定される時は全てエラーメッセージが出力されるようにしてください

# Chapter VIII

## Exercise 05 : a slice of ...

	Exercise 05
	a slice of ...
	提出するディレクトリ : <i>ex05/</i>
	提出するファイル : <i>subSlice.go</i>
	使用可能なパッケージ : <i>fmt</i>

- 引数にint型のスライスとint型の値begin, length, capacityをとり、新しいint型のスライスを返すsubSlice関数を作ってください。

```
func subSlice(slice []int, begin int, length int, capacity int) []int {  
    [...]  
}
```

- 新しいスライスの各要素は、元のスライスのbegin番目からlength分コピーしたものとし、足りない分は0で埋めてください。
- スライスの容量はcapacityで指定された容量としますが、capacityがlengthより小さい場合はlengthを優先してください。
- 以下はmain関数と標準出力の例です。


```
func main() {  
    var orig = []int{0, 1, 2, 3, 4, 5}  
    var ret []int  
  
    ret = subSlice(orig, 0, 3, 3)  
    fmt.Printf("ret = %v, len = %d, cap = %d\n", ret, len(ret), cap(ret))  
  
    ret = subSlice(orig, 2, 7, 10)  
    fmt.Printf("ret = %v, len = %d, cap = %d\n", ret, len(ret), cap(ret))  
}
```

```
ret = subSlice(orig, 2, 7, 3)
fmt.Printf("ret = %v, len = %d, cap = %d\n", ret, len(ret), cap(ret))
}
```

```
$ go build subSlice.go
$ ./subSlice | cat -e
ret = [0 1 2], len = 3, cap = 3$
ret = [2 3 4 5 0 0 0], len = 7, cap = 10$
ret = [2 3 4 5 0 0 0], len = 7, cap = 7$
$
```

# Chapter IX

## Exercise 06 : echo!!!

	Exercise 06
echo!!!	
提出するディレクトリ : <i>ex06/</i>	
提出するファイル : <i>echo42.go</i>	
使用可能なパッケージ : <i>flag, fmt, strings</i>	

- echo42コマンドを実装しなさい。echo42コマンドはnとsで表されるoptionを受け取ります。出力はすべて標準出力です。
- -nオプションは最後の改行を省略します。
- -sオプションは空白ではなく、-sの次に渡される文字列の内容で出力引数を区切って出力します。

```
$ go build echo42.go
$ ./echo42 12 34 555 | cat -e
12 34 555$
$ ./echo42 -s / a bc def | cat -e
a/bc/def$
$ ./echo42 -s XXXXXX 12 34 56789 | cat -e
12XXXXXXX34XXXXXXX56789$
$ ./echo42 -n 12 34 555 | cat -e
12 34 555
$ ./echo42 -help
Usage of ./echo42:
  -n omit trailing newline
  -s string
      separator (default " ")
$ ./echo42 | cat -e
$ $
$
```

- -nオプションをつけると最後の改行が省略されていることに注意。



flag??