

《Go 每日一题》一共 120 天，每天 2-3 题。由 SeekLoad 整理，公众号：Golang来啦。



由「Go语言中文网」同步发布：



第 1 天

1.下面这段代码输出的内容

```
1 package main
2
3 import (
4     "fmt"
5 )
6
7 func main() {
8     defer_call()
9 }
10
11 func defer_call() {
12     defer func() { fmt.Println("打印前") }()
```

```
13     defer func() { fmt.Println("打印中") }()
14     defer func() { fmt.Println("打印后") }()
15
16     panic("触发异常")
17 }
```

答案:

```
1 打印后
2 打印中
3 打印前
4 panic: 触发异常
```

解析: defer 的执行顺序是后进先出。当出现 panic 语句的时候, 会先按照 defer 的后进先出的顺序执行, 最后才会执行panic。

第 2 天

1.下面这段代码输出什么, 说明原因。

```
1 func main() {
2
3     slice := []int{0,1,2,3}
4     m := make(map[int]*int)
5
6     for key,val := range slice {
7         m[key] = &val
8     }
9
10    for k,v := range m {
11        fmt.Println(k,"->",*v)
12    }
13 }
14
```

参考答案:

```
1 0 -> 3
2 1 -> 3
3 2 -> 3
4 3 -> 3
```

解析: 这是新手常会犯的错误写法, for range 循环的时候会创建每个元素的副本, 而不是元素的引用, 所以 m[key] = &val 取的都是变量 val 的地址, 所以最后 map 中的所有元素的值都是变量 val 的地址, 因为最后 val 被赋值为3, 所有输出都是3。

正确的写法:

```

1 func main() {
2
3     slice := []int{0,1,2,3}
4     m := make(map[int]*int)
5
6     for key,val := range slice {
7         value := val
8         m[key] = &value
9     }
10
11    for k,v := range m {
12        fmt.Println(k,"==>",*v)
13    }
14 }

```

扩展题目

```

1 type Test struct {
2     name string
3 }
4
5 func (this *Test) Point(){
6     fmt.Println(this.name)
7 }
8
9
10 func main() {
11
12     ts := []Test{
13         {"a"},
14         {"b"},
15         {"c"},
16     }
17
18     for _,t := range ts {
19         //fmt.Println(reflect.TypeOf(t))
20         defer t.Point()
21     }
22
23 }

```

参考: <https://blog.csdn.net/idwtwt/article/details/87378419>

第 3 天

1.下面两段代码输出什么。

```
1 // 1.
2 func main() {
3     s := make([]int, 5)
4     s = append(s, 1, 2, 3)
5     fmt.Println(s)
6 }
7
8 // 2.
9 func main() {
10    s := make([]int,0)
11    s = append(s,1,2,3,4)
12    fmt.Println(s)
13 }
```

2.下面这段代码有什么缺陷

```
1 func funcMui(x,y int)(sum int,error){
2     return x+y,nil
3 }
```

答案：第二个返回值没有命名。解析：在函数有多个返回值时，只要有一个返回值有命名，其他的也必须命名。如果有多个返回值必须加上括号()；如果只有一个返回值且命名也必须加上括号()。这里的第一个返回值有命名 sum，第二个没有命名，所以错误。

3.new() 与 make() 的区别。

new(T) 和 make(T,args) 是 Go 语言内建函数，用来分配内存，但适用的类型不同。

new(T) 会为 T 类型的新值分配已置零的内存空间，并返回地址（指针），即类型为 *T 的值。换句话说就是，返回一个指针，该指针指向新分配的、类型为 T 的零值。适用于值类型，如数组、结构体等。

make(T,args) 返回初始化之后的 T 类型的值，这个值并不是 T 类型的零值，也不是指针 *T，是经过初始化之后的 T 的引用。make() 只适用于 slice、map 和 channel。

第 4 天

1.下面这段代码能否通过编译，不能的话原因是什么；如果通过，输出什么。

```
1 func main() {
2     list := new([]int)
3     list = append(list, 1)
4     fmt.Println(list)
5 }
```

参考答案：不能通过编译，new([]int) 之后的 list 是一个 *[]int 类型的指针，不能对指针执行 append 操作。可以使用 make() 初始化之后再使用。同样的，map 和 channel 建议使用 make() 或字面量的方式初始化，不要用 new()。

2.下面这段代码能否通过编译，如果可以，输出什么？

```
1 func main() {
2     s1 := []int{1, 2, 3}
3     s2 := []int{4, 5}
4     s1 = append(s1, s2)
5     fmt.Println(s1)
6 }
```

参考答案：不能通过编译。append() 的第二个参数不能直接使用 slice，需使用 ... 操作符，将一个切片追加到另一个切片上：append(s1,s2...)。或者直接跟上元素，形如：append(s1,1,2,3)。

3.下面这段代码能否通过编译，如果可以，输出什么？

```
1 var(
2     size := 1024
3     max_size = size*2
4 )
5
6 func main() {
7     fmt.Println(size,max_size)
8 }
```

参考答案：不能通过编译。

参考解析：这道题的主要知识点是变量声明的简短模式，形如：x := 100. 但这种声明方式有限制：

1. 必须使用显示初始化；
2. 不能提供数据类型，编译器会自动推导；
3. 只能在函数内部使用简短模式；

第 5 天

1.下面这段代码能否通过编译？不能的话，原因是什么？如果通过，输出什么？

```
1 func main() {
2     sn1 := struct {
3         age int
4         name string
5     }{age: 11, name: "qq"}
6     sn2 := struct {
7         age int
8         name string
9     }{age: 11, name: "qq"}
10 }
```

```

11     if sn1 == sn2 {
12         fmt.Println("sn1 == sn2")
13     }
14
15     sm1 := struct {
16         age int
17         m   map[string]string
18     }{age: 11, m: map[string]string{"a": "1"}}
19     sm2 := struct {
20         age int
21         m   map[string]string
22     }{age: 11, m: map[string]string{"a": "1"}}
23
24     if sm1 == sm2 {
25         fmt.Println("sm1 == sm2")
26     }
27 }

```

参考答案：编译不通过 invalid operation: sm1 == sm2

参考解析：这道题目考的是结构体的比较：

1. 结构体只能比较是否相等，但是不能比较大小。
2. 相同类型的结构体才能够进行比较，结构体是否相同不但与属性类型有关，还与属性顺序相关，sn3 与 sn1 就是不同的结构体；

```

1     sn3:= struct {
2         name string
3         age  int
4     }{age:11,name:"qq"}

```

3. 如果 struct 的所有成员都可以比较，则该 struct 就可以通过 == 或 != 进行比较是否相等，比较时逐个项进行比较，如果每一项都相等，则两个结构体才相等，否则不相等；

那什么是可比较的呢，常见的有 bool、数值型、string、指针、数组等，像切片、map、函数等是不能比较的。具体可以参考 Go 说明文档。https://golang.org/ref/spec#Comparison_operators

第 6 天

1.通过指针变量 p 访问其成员变量 name，有哪几种方式？

- A.p.name
- B.&p.name
- C.(*p).name
- D.p->name

参考答案：AC 参考解析：& 取址运算符，* 指针解引用。

2.下面这段代码能否通过编译？如果通过，输出什么？

```
1 package main
2
3 import "fmt"
4
5 type MyInt1 int
6 type MyInt2 = int
7
8 func main() {
9     var i int = 0
10    var i1 MyInt1 = i
11    var i2 MyInt2 = i
12    fmt.Println(i1,i2)
13 }
```

参考答案：编译不通过，cannot use i (type int) as type MyInt1 in assignment

参考解析：这道题考的是类型别名与类型定义的区别。

第 5 行代码是基于类型 int 创建了新类型 MyInt1，第 6 行代码是创建了 int 的类型别名 MyInt2，注意类型别名的定义时 = 。所以，第 10 行代码相当于是将 int 类型的变量赋值给 MyInt1 类型的变量，Go 是强类型语言，编译当然不通过；而 MyInt2 只是 int 的别名，本质上还是 int，可以赋值。

第 10 行代码的赋值可以使用强制类型转化 var i1 MyInt1 = MyInt1(i)。

第 7 天

1.关于字符串连接，下面语法正确的是？

- A. str := 'abc' + '123'
- B. str := "abc" + "123"
- C. str := '123' + "abc"
- S. fmt.Sprintf("abc%d", 123)

参考答案：BD

参考解析：考的知识点是字符串连接。除了以上两种连接方式，还有 strings.Join()、buffer.WriteString()等。

2.下面这段代码能否编译通过？如果可以，输出什么？

```

1  const (
2      x = iota
3      _
4      y
5      z = "zz"
6      k
7      p = iota
8  )
9
10 func main() {
11     fmt.Println(x,y,z,k,p)
12 }

```

参考答案：编译通过，输出：0 2 zz zz 5

参考解析：知识点 iota。参考 <https://www.cnblogs.com/zsy/p/5370052.html>

3.下面赋值正确的是()

- A. var x = nil
- B. var x interface{} = nil
- C. var x string = nil
- D. var x error = nil

参考答案及解析：BD。这道题考的知识点是 nil。nil 只能赋值给指针、chan、func、interface、map 或 slice 类型的变量。强调下 D 选项的 error 类型，它是一种内置接口类型，看它的源码就知道，所以 D 是对的。

```

1  type error interface {
2      Error() string
3  }

```

第 8 天

1.关于init函数，下面说法正确的是()

- A. 一个包中，可以包含多个 init 函数；
- B. 程序编译时，先执行依赖包的 init 函数，再执行 main 包内的 init 函数；
- C. main 包中，不能有 init 函数；
- D. init 函数可以被其他函数调用；

参考答案及解析：AB。关于 init() 函数有几个需要注意的地方：

1. init() 函数是用于程序执行前做包的初始化的函数，比如初始化包里的变量等；
2. 一个包可以出线多个 init() 函数,一个源文件也可以包含多个 init() 函数；
3. 同一个包中多个 init() 函数的执行顺序没有明确定义，**但是不同包的init函数是根据包导入的依赖关系决定的**；
4. init() 函数在代码中不能被显示调用、不能被引用（赋值给函数变量），否则出现编译错误；
5. 一个包被引用多次，如 A import B,C import B,A import C，B 被引用多次，但 B 包只会初始化一次；
6. 引入包，不可出现死循环。即A import B,B import A，这种情况编译失败；

2.下面这段代码输出什么以及原因？

```

1 func hello() []string {
2     return nil
3 }
4
5 func main() {
6     h := hello
7     if h == nil {
8         fmt.Println("nil")
9     } else {
10        fmt.Println("not nil")
11    }
12 }

```

- A. nil
- B. not nil
- C. compilation error

答案及解析：B。这道题目里面，是将 hello() 赋值给变量 h，而不是函数的返回值，所以输出 not nil。

3.下面这段代码能否编译通过？如果可以，输出什么？

```

1 func GetValue() int {
2     return 1
3 }
4
5 func main() {
6     i := GetValue()
7     switch i.(type) {
8     case int:
9         println("int")
10    case string:
11        println("string")
12    case interface{}:
13        println("interface")
14    default:
15        println("unknown")
16    }
17 }

```

答案及解析：编译失败。考点：类型选择，类型选择的语法形如：i.(type)，其中 i 是接口，type 是固定关键字，需要注意的是，只有接口类型才可以使用类型选择。

第 9 天

1.关于 channel，下面语法正确的是（）

- A. var ch chan int
- B. ch := make(chan int)
- C. <- ch

- D. ch <-

参考答案及解析：ABC。A、B都是声明 channel；C 读取 channel；写 channel 是必须带上值，所以 D 错误。

2.下面这段代码输出什么？

- A.0
- B.1
- C.Compilation error

```
1 type person struct {
2     name string
3 }
4
5 func main() {
6     var m map[person]int
7     p := person{"mike"}
8     fmt.Println(m[p])
9 }
```

参考答案及解析：A。打印一个 map 中不存在的值时，返回元素类型的零值。这个例子中，m 的类型是 map[person]int，因为 m 中不存在 p，所以打印 int 类型的零值，即 0。

3.下面这段代码输出什么？

- A.18
- B.5
- C.Compilation error

```
1 func hello(num ...int) {
2     num[0] = 18
3 }
4
5 func main() {
6     i := []int{5, 6, 7}
7     hello(i...)
8     fmt.Println(i[0])
9 }
```

参考答案及解析：18。知识点：可变函数。可以看下之前发过的文章。

第 10 天

1.下面这段代码输出什么？

```
1 func main() {
2     a := 5
3     b := 8.1
4     fmt.Println(a + b)
5 }
```

- A.13.1
- B.13
- C.compilation error

参考答案及解析：C。a 的类型是 int，b 的类型是 float，两个不同类型的数值不能相加，编译报错。

2.下面这段代码输出什么？

```
1 package main
2
3 import (
4     "fmt"
5 )
6
7 func main() {
8     a := [5]int{1, 2, 3, 4, 5}
9     t := a[3:4:4]
10    fmt.Println(t[0])
11 }
```

- A.3
- B.4
- C.compilation error

参考答案及解析：B。

知识点：操作符 [i,j]。基于数组（切片）可以使用操作符 [i,j] 创建新的切片，从索引 i，到索引 j 结束，截取已有数组（切片）的任意部分，返回新的切片，新切片的值包含原数组（切片）的 i 索引的值，但是不包含 j 索引的值。i、j 都是可选的，i 如果省略，默认是 0，j 如果省略，默认是原数组（切片）的长度。i、j 都不能超过这个长度值。

假如底层数组的大小为 k，截取之后获得的切片的长度和容量的计算方法：

长度：j-i，容量：k-i

截取操作符还可以有第三个参数，形如 [i,j,k]，第三个参数 k 用来限制新切片的容量，但不能超过原数组（切片）的底层数组大小。截取获得的切片的长度和容量分别是：**j-i、k-i**。

所以例子中，切片 t 为 [4]，长度和容量都是 1。

3.下面这段代码输出什么？

```
1 func main() {
2     a := [2]int{5, 6}
3     b := [3]int{5, 6}
4     if a == b {
5         fmt.Println("equal")
6     } else {
7         fmt.Println("not equal")
8     }
9 }
```

- A. compilation error
- B. equal
- C. not equal

参考答案及解析：A。Go 中的数组是值类型，可比较，另外一方面，数组的长度也是数组类型的组成部分，所以 a 和 b 是不同的类型，是不能比较的，所以编译错误。

第 11 天

1.关于 cap() 函数的适用类型，下面说法正确的是()

- A. array
- B. slice
- C. map
- D. channel

参考答案及解析：ABD。知识点：cap(), cap() 函数不适用 map。

2.下面这段代码输出什么？

```
1 func main() {
2     var i interface{}
3     if i == nil {
4         fmt.Println("nil")
5         return
6     }
7     fmt.Println("not nil")
8 }
```

- A. nil
- B. not nil
- C. compilation error

参考答案及解析：A。当且仅当接口的动态值和动态类型都为 nil 时，接口类型值才为 nil。

3.下面这段代码输出什么？

```
1 func main() {
2     s := make(map[string]int)
3     delete(s, "h")
4     fmt.Println(s["h"])
5 }
```

- A. runtime panic
- B. 0
- C. compilation error

参考答案及解析：B。删除 map 不存在的键值对时，不会报错，相当于没有任何作用；获取不存在的键值对时，返回值类型对应的零值，所以返回 0。

第 12 天

1. 下面属于关键字的是()

- A. func
- B. struct
- C. class
- D. defer

参考答案及解析：ABD。知识点：Go 语言的 25 个关键字。

2. 下面这段代码输出什么？

```
1 func main() {
2     i := -5
3     j := +5
4     fmt.Printf("%+d %+d", i, j)
5 }
```

- A. -5 +5
- B. +5 +5
- C. 0 0

参考答案及解析：A。%d 表示输出十进制数字，+ 表示输出数值的符号。这里不表示取反。

3. 下面这段代码输出什么？

```
1 type People struct{}
2
3 func (p *People) ShowA() {
4     fmt.Println("showA")
5     p.ShowB()
6 }
7 func (p *People) ShowB() {
8     fmt.Println("showB")
```

```

9   }
10
11  type Teacher struct {
12      People
13  }
14
15  func (t *Teacher) ShowB() {
16      fmt.Println("teacher showB")
17  }
18
19  func main() {
20      t := Teacher{}
21      t.ShowB()
22  }

```

参考答案及解析：teacher showB。知识点：结构体嵌套。在嵌套结构体中，People 称为内部类型，Teacher 称为外部类型；通过嵌套，内部类型的属性、方法，可以为外部类型所有，就好像是外部类型自己的一样。此外，外部类型还可以定义自己的属性和方法，甚至可以定义与内部相同的方法，这样内部类型的方法就会被“屏蔽”。这个例子中的 ShowB() 就是同名方法。

第 13 天

1. 定义一个包内全局字符串变量，下面语法正确的是（）

- A. var str string
- B. str := ""
- C. str = ""
- D. var str = ""

参考答案及解析：AD。B 只支持局部变量声明；C 是赋值，str 必须在这之前已经声明；

2. 下面这段代码输出什么？

```

1  func hello(i int) {
2      fmt.Println(i)
3  }
4  func main() {
5      i := 5
6      defer hello(i)
7      i = i + 10
8  }

```

参考答案及解析：5。这个例子中，hello() 函数的参数在执行 defer 语句的时候会保存一份副本，在实际调用 hello() 函数时用，所以是 5。

3. 下面这段代码输出什么？

```

1  type People struct{}
2

```

```

3 func (p *People) ShowA() {
4     fmt.Println("showA")
5     p.ShowB()
6 }
7 func (p *People) ShowB() {
8     fmt.Println("showB")
9 }
10
11 type Teacher struct {
12     People
13 }
14
15 func (t *Teacher) ShowB() {
16     fmt.Println("teacher showB")
17 }
18
19 func main() {
20     t := Teacher{}
21     t.ShowA()
22 }

```

参考答案及解析：

```

1 showA
2 showB

```

知识点：结构体嵌套。这道题可以结合第 12 天的第三题一起看，Teacher 没有自己 ShowA()，所以调用内部类型 People 的同名方法，需要注意的是第 5 行代码调用的是 People 自己的 ShowB 方法。

第 14 天

1. 下列选项正确的是？

```

1 func main() {
2     str := "hello"
3     str[0] = 'x'
4     fmt.Println(str)
5 }

```

- A. hello
- B. xello
- C. compilation error

参考代码及解析：C。知识点：常量，Go 语言中的字符串是只读的。

2. 下面代码输出什么？

```

1 func incr(p *int) int {
2     *p++
3     return *p
4 }
5
6 func main() {
7     p := 1
8     incr(&p)
9     fmt.Println(p)
10 }

```

- A. 1
- B. 2
- C. 3

参考答案及解析：B。知识点：指针，incr() 函数里的 p 是 `*int` 类型的指针，指向的是 main() 函数的变量 p 的地址。第 2 行代码是将该地址的值执行一个自增操作，incr() 返回自增后的结果。

3.对 add() 函数调用正确的是？

```

1 func add(args ...int) int {
2
3     sum := 0
4     for _, arg := range args {
5         sum += arg
6     }
7     return sum
8 }

```

- A. add(1, 2)
- B. add(1, 3, 7)
- C. add([]int{1, 2})
- D. add([]int{1, 3, 7}...)

参考答案及解析：ABD。知识点：可变函数。

第 15 天

1.下面代码下划线处可以填入哪个选项？

```
1 func main() {
2     var s1 []int
3     var s2 = []int{}
4     if __ == nil {
5         fmt.Println("yes nil")
6     }else{
7         fmt.Println("no nil")
8     }
9 }
```

- A. s1
- B. s2
- C. s1、s2 都可以

参考答案及解析：A。知识点：nil 切片和空切片。nil 切片和 nil 相等，一般用来表示一个不存在的切片；空切片和 nil 不相等，表示一个空的集合。

2.下面这段代码输出什么？

```
1 func main() {
2     i := 65
3     fmt.Println(string(i))
4 }
```

- A. A
- B. 65
- C. compilation error

参考答案及解析：A。UTF-8 编码中，十进制数字 65 对应的符号是 A。

3.下面这段代码输出什么？

```
1 type A interface {
2     ShowA() int
3 }
4
5 type B interface {
6     ShowB() int
7 }
8
9 type Work struct {
10    i int
11 }
12
13 func (w Work) ShowA() int {
14    return w.i + 10
15 }
16
17 func (w Work) ShowB() int {
```

```

18     return w.i + 20
19 }
20
21 func main() {
22     c := Work{3}
23     var a A = c
24     var b B = c
25     fmt.Println(a.ShowA())
26     fmt.Println(b.ShowB())
27 }

```

参考答案及解析：13 23。知识点：接口。一种类型实现多个接口，结构体 Work 分别实现了接口 A、B，所以接口变量 a、b 调用各自的方法 ShowA() 和 ShowB()，输出 13、23。

第 16 天

1.切片 a、b、c 的长度和容量分别是多少？

```

1 func main() {
2
3     s := [3]int{1, 2, 3}
4     a := s[:0]
5     b := s[:2]
6     c := s[1:2:cap(s)]
7 }

```

参考答案及解析：a、b、c 的长度和容量分别是 0 3、2 3、1 2。知识点：数组或切片的截取操作。截取操作有带 2 个或者 3 个参数，形如：[i:j] 和 [i:j:k]，假设截取对象的底层数组长度为 l。在操作符 [i:j] 中，如果 i 省略，默认 0，如果 j 省略，默认底层数组的长度，截取得到的切片长度和容量计算方法是 j-i、l-i。操作符 [i:j:k]，k 主要是用来限制切片的容量，但是不能大于数组的长度 l，截取得到的切片长度和容量计算方法是 j-i、k-i。

2.下面代码中 A B 两处应该怎么修改才能顺利编译？

```

1 func main() {
2     var m map[string]int //A
3     m["a"] = 1
4     if v := m["b"]; v != nil { //B
5         fmt.Println(v)
6     }
7 }

```

参考答案及解析：

```

1 func main() {
2     m := make(map[string]int)
3     m["a"] = 1
4     if v,ok := m["b"]; ok {
5         fmt.Println(v)
6     }
7 }

```

在 A 处只声明了 map m ,并没有分配内存空间，不能直接赋值，需要使用 make()，都提倡使用 make() 或者字面量的方式直接初始化 map。

B 处，`v,k := m["b"]` 当 key 为 b 的元素不存在的时候，v 会返回值类型对应的零值，k 返回 false。

3.下面代码输出什么？

```

1 type A interface {
2     ShowA() int
3 }
4
5 type B interface {
6     ShowB() int
7 }
8
9 type Work struct {
10     i int
11 }
12
13 func (w Work) ShowA() int {
14     return w.i + 10
15 }
16
17 func (w Work) ShowB() int {
18     return w.i + 20
19 }
20
21 func main() {
22     c := Work{3}
23     var a A = c
24     var b B = c
25     fmt.Println(a.ShowB())
26     fmt.Println(b.ShowA())
27 }

```

- A. 23 13
- B. compilation error

参考答案及解析：B。知识点：接口的静态类型。a、b 具有相同的动态类型和动态值，分别是结构体 work 和 {3}；a 的静态类型是 A，b 的静态类型是 B，接口 A 不包括方法 ShowB()，接口 B 也不包括方法 ShowA()，编译报错。看下编译错误：

```
1 a.ShowB undefined (type A has no field or method ShowB)
2 b.ShowA undefined (type B has no field or method ShowA)
```

第 17 天

1.下面代码中，x 已声明，y 没有声明，判断每条语句的对错。

```
1 1. x, _ := f()
2 2. x, _ = f()
3 3. x, y := f()
4 4. x, y = f()
```

参考答案及解析：错、对、对、错。知识点：变量的声明。1.错，x 已经声明，不能使用 :=；2.对；3.对，当多值赋值时，:= 左边的变量无论声明与否都可以；4.错，y 没有声明。

2.下面代码输出什么？

```
1 func increaseA() int {
2     var i int
3     defer func() {
4         i++
5     }()
6     return i
7 }
8
9 func increaseB() (r int) {
10    defer func() {
11        r++
12    }()
13    return r
14 }
15
16 func main() {
17    fmt.Println(increaseA())
18    fmt.Println(increaseB())
19 }
```

- A. 1 1
- B. 0 1
- C. 1 0
- D. 0 0

参考答案及解析：B。知识点：defer、返回值。注意一下，increaseA() 的返回参数是匿名，increaseB() 是具名。关于 defer 与返回值的知识点，后面我会写篇文章详细分析，到时候可以看下文章的讲解。

3.下面代码输出什么？

```
1 type A interface {
2     ShowA() int
3 }
4
5 type B interface {
6     ShowB() int
7 }
8
9 type Work struct {
10    i int
11 }
12
13 func (w Work) ShowA() int {
14    return w.i + 10
15 }
16
17 func (w Work) ShowB() int {
18    return w.i + 20
19 }
20
21 func main() {
22    var a A = Work{3}
23    s := a.(Work)
24    fmt.Println(s.ShowA())
25    fmt.Println(s.ShowB())
26 }
```

- A. 13 23
- B. compilation error

参考答案及解析：A。知识点：类型断言。这道题可以和第 15 天的第三题 和第 16 天的第三题结合起来看。

第 18 天

1.f1()、f2()、f3() 函数分别返回什么？

```
1 func f1() (r int) {
2     defer func() {
3         r++
4     }()
5     return 0
6 }
7
8
9 func f2() (r int) {
10    t := 5
11    defer func() {
```

```

12     t = t + 5
13 }()
14 return t
15 }
16
17
18 func f3() (r int) {
19     defer func(r int) {
20         r = r + 5
21     }(r)
22     return 1
23 }

```

参考答案及解析：1 5 1。知识点：defer、返回值。

第 19 天

1.下面代码段输出什么？

```

1 type Person struct {
2     age int
3 }
4
5 func main() {
6     person := &Person{28}
7
8     // 1.
9     defer fmt.Println(person.age)
10
11    // 2.
12    defer func(p *Person) {
13        fmt.Println(p.age)
14    }(person)
15
16    // 3.
17    defer func() {
18        fmt.Println(person.age)
19    }()
20
21    person.age = 29
22 }

```

参考答案及解析：29 29 28。变量 person 是一个指针变量。

1.person.age 此时是将 28 当做 defer 函数的参数，会把 28 缓存在栈中，等到最后执行该 defer 语句的时候取出，即输出 28；

2.defer 缓存的是结构体 Person{28} 的地址，最终 Person{28} 的 age 被重新赋值为 29，所以 defer 语句最后执行的时候，依靠缓存的地址取出的 age 便是 29，即输出 29；

3.很简单，闭包引用，输出 29；

又由于 defer 的执行顺序为先进后出，即 3 2 1，所以输出 29 29 28。

第 20 天

1.下面这段代码正确的输出是什么？

```
1 func f() {
2     defer fmt.Println("D")
3     fmt.Println("F")
4 }
5
6 func main() {
7     f()
8     fmt.Println("M")
9 }
```

- A. F M D
- B. D F M
- C. F D M

参考答案及解析：C。被调用函数里的 defer 语句在返回之前就会被执行，所以输出顺序是 F D M。

2.下面代码输出什么？

```
1 type Person struct {
2     age int
3 }
4
5 func main() {
6     person := &Person{28}
7
8     // 1.
9     defer fmt.Println(person.age)
10
11    // 2.
12    defer func(p *Person) {
13        fmt.Println(p.age)
14    }(person)
15
16    // 3.
17    defer func() {
18        fmt.Println(person.age)
19    }()
20
21    person = &Person{29}
22 }
```

参考答案及解析：29 28 28。

这道题在第 19 天题目的基础上做了一点点小改动，前一题最后一行代码 `person.age = 29` 是修改引用对象的成员 `age`，这题最后一行代码 `person = &Person{29}` 是修改引用对象本身，来看看有什么区别。

1. `person.age` 这一行代码跟之前含义是一样的，此时是将 28 当做 `defer` 函数的参数，会把 28 缓存在栈中，等到最后执行该 `defer` 语句的时候取出，即输出 28；

2. `defer` 缓存的是结构体 `Person{28}` 的地址，这个地址指向的结构体没有被改变，最后 `defer` 语句后面的函数执行的时候取出仍是 28；

3. 闭包引用，`person` 的值已经被改变，指向结构体 `Person{29}`，所以输出 29。

由于 `defer` 的执行顺序为先进后出，即 3 2 1，所以输出 29 28 28。

第 21 天

1. 下面的两个切片声明中有什么区别？哪个更可取？

```
1 | A. var a []int
2 | B. a := []int{}
```

参考答案及解析：第一种切片声明不会分配内存，优先选择。

2. A、B、C、D 哪些选项有语法错误？

```
1 | type S struct {
2 | }
3 |
4 | func f(x interface{}) {
5 | }
6 |
7 | func g(x *interface{}) {
8 | }
9 |
10 | func main() {
11 |     s := S{}
12 |     p := &s
13 |     f(s) //A
14 |     g(s) //B
15 |     f(p) //C
16 |     g(p) //D
17 | }
```

参考答案及解析：BD。函数参数为 `interface{}` 时可以接收任何类型的参数，包括用户自定义类型等，即使是接收指针类型也用 `interface{}`，而不是使用 `*interface{}`。

永远不要使用一个指针指向一个接口类型，因为它已经是一个指针。

3. 下面 A、B 两处应该填入什么代码，才能确保顺利打印出结果？

```

1  type S struct {
2      m string
3  }
4
5  func f() *S {
6      return __ //A
7  }
8
9  func main() {
10     p := __ //B
11     fmt.Println(p.m) //print "foo"
12 }

```

参考答案及解析：

```

1  A. &S{"foo"}
2  B. *f() 或者 f()

```

f() 函数返回参数是指针类型，所以可以用 & 取结构体的指针；B 处，如果填 `*f()`，则 p 是 S 类型；如果填 `f()`，则 p 是 *S 类型，不过都可以使用 `p.m` 取得结构体的成员。

第 22 天

1. 下面的代码有几处语法问题，各是什么？

```

1  package main
2  import (
3      "fmt"
4  )
5  func main() {
6      var x string = nil
7      if x == nil {
8          x = "default"
9      }
10     fmt.Println(x)
11 }

```

参考答案及解析：2 处有语法问题。golang 的字符串类型是不能赋值 nil 的，也不能跟 nil 比较。

2. return 之后的 defer 语句会执行吗，下面这段代码输出什么？

```

1  var a bool = true
2  func main() {
3      defer func(){
4          fmt.Println("1")
5      }()
6      if a == true {
7          fmt.Println("2")
8          return
9      }
10     defer func(){
11         fmt.Println("3")
12     }()
13 }

```

参考答案及解析：2 1。defer 关键字后面的函数或者方法想要执行必须先注册，return 之后的 defer 是不能注册的，也就不能执行后面的函数或方法。

Reference:

1. <https://studygolang.com/topics/9967>

第 23 天

1.下面这段代码输出什么？为什么？

```

1  func main() {
2
3      s1 := []int{1, 2, 3}
4      s2 := s1[1:]
5      s2[1] = 4
6      fmt.Println(s1)
7      s2 = append(s2, 5, 6, 7)
8      fmt.Println(s1)
9  }

```

参考答案及解析：

```

1  [1 2 4]
2  [1 2 4]

```

我们已经知道，golang 中切片底层的数据结构是数组。当使用 `s1[1:]` 获得切片 `s2`，和 `s1` 共享同一个底层数组，这会导致 `s2[1] = 4` 语句影响 `s1`。

而 `append` 操作会导致底层数组扩容，生成新的数组，因此追加数据后的 `s2` 不会影响 `s1`。

但是为什么对 `s2` 赋值后影响的却是 `s1` 的第三个元素呢？这是因为切片 `s2` 是从数组的第二个元素开始，`s2` 索引为 1 的元素对应的是 `s1` 索引为 2 的元素。

2.下面选项正确的是？

```
1 func main() {
2     if a := 1; false {
3     } else if b := 2; false {
4     } else {
5         println(a, b)
6     }
7 }
```

- A. 1 2
- B. compilation error

参考答案及解析：A。

推荐一篇文章，讲的很详细 <https://tonybai.com/2018/05/11/the-analysis-of-a-go-code-snippet-about-code-blocks-and-scope/>

第 24 天

1.下面这段代码输出什么？

```
1 func main() {
2     m := map[int]string{0:"zero",1:"one"}
3     for k,v := range m {
4         fmt.Println(k,v)
5     }
6 }
```

参考答案及解析：

```
1 0 zero
2 1 one
3 // 或者
4 1 one
5 0 zero
```

map 的输出是无序的。

2.下面这段代码输出什么？

```
1 func main() {
2     a := 1
3     b := 2
4     defer calc("1", a, calc("10", a, b))
5     a = 0
6     defer calc("2", a, calc("20", a, b))
7     b = 1
8 }
```

```

9
10 func calc(index string, a, b int) int {
11     ret := a + b
12     fmt.Println(index, a, b, ret)
13     return ret
14 }

```

参考答案及解析：

```

1 10 1 2 3
2 20 0 2 2
3 2 0 2 2
4 1 1 3 4

```

程序执行到 main() 函数三行代码的时候，会先执行 calc() 函数的 b 参数，即：calc("10",a,b)，输出：10 1 2 3，得到值 3，因为 defer 定义的函数是延迟函数，故 calc("1",1,3) 会被延迟执行；

程序执行到第五行的时候，同样先执行 calc("20",a,b) 输出：20 0 2 2 得到值 2，同样将 calc("2",0,2) 延迟执行；

程序执行到末尾的时候，按照栈先进后出的方式依次执行：calc("2",0,2)，calc("1",1,3)，则就依次输出：2 0 2 2，1 1 3 4。

第 25 天

1.下面这段代码输出什么？为什么？

```

1 func (i int) PrintInt () {
2     fmt.Println(i)
3 }
4
5 func main() {
6     var i int = 1
7     i.PrintInt()
8 }

```

- A. 1
- B. compilation error

参考答案及解析：B。基于类型创建的方法必须定义在同一个包内，上面的代码基于 int 类型创建了 PrintInt() 方法，由于 int 类型和方法 PrintInt() 定义在不同的包内，所以编译出错。解决的办法可以定义一种新的类型：

```

1  type Myint int
2
3  func (i Myint) PrintInt () {
4      fmt.Println(i)
5  }
6
7  func main() {
8      var i Myint = 1
9      i.PrintInt()
10 }

```

2.下面这段代码输出什么? 为什么?

```

1  type People interface {
2      Speak(string) string
3  }
4
5  type Student struct{}
6
7  func (stu *Student) Speak(think string) (talk string) {
8      if think == "speak" {
9          talk = "speak"
10     } else {
11         talk = "hi"
12     }
13     return
14 }
15
16 func main() {
17     var peo People = Student{}
18     think := "speak"
19     fmt.Println(peo.Speak(think))
20 }

```

- A. speak
- B. compilation error

参考答案及解析: B。编译错误 `Student does not implement People (Speak method has pointer receiver)`, 值类型 `Student` 没有实现接口的 `Speak()` 方法, 而是指针类型 `*Student` 实现该方法。

详细请参考这篇文章 <https://seekload.net/2019/06/06/go-study-method.html>

第 26 天

1.下面这段代码输出什么?

```

1  const (
2      a = iota

```

```

3   b = iota
4   )
5   const (
6     name = "name"
7     c     = iota
8     d     = iota
9   )
10  func main() {
11    fmt.Println(a)
12    fmt.Println(b)
13    fmt.Println(c)
14    fmt.Println(d)
15  }

```

参考答案及解析：0 1 1 2。知识点：iota 的用法。

iota 是 go 语言的常量计数器，只能在常量的表达式中使用。

iota 在 const 关键字出现时将被重置为 0，const 中每新增一行常量声明将使 iota 计数一次。

Reference:

1. <https://studygolang.com/articles/2192>

2. 下面这段代码输出什么？为什么？

```

1   type People interface {
2     Show()
3   }
4
5   type Student struct{}
6
7   func (stu *Student) Show() {
8
9   }
10
11  func main() {
12
13    var s *Student
14    if s == nil {
15      fmt.Println("s is nil")
16    } else {
17      fmt.Println("s is not nil")
18    }
19    var p People = s
20    if p == nil {
21      fmt.Println("p is nil")
22    } else {

```

```
23     fmt.Println("p is not nil")
24 }
25 }
```

参考答案及解析：`s is nil` 和 `p is not nil`。这道题会不会有点诧异，我们分配给变量 `p` 的值明明是 `nil`，然而 `p` 却不是 `nil`。记住一点，当且仅当动态值和动态类型都为 `nil` 时，接口类型值才为 `nil`。上面的代码，给变量 `p` 赋值之后，`p` 的动态值是 `nil`，但是动态类型却是 `*Student`，是一个 `nil` 指针，所以相等条件不成立。

第 27 天

1.下面这段代码输出什么？

```
1  type Direction int
2
3  const (
4      North Direction = iota
5      East
6      South
7      West
8  )
9
10 func (d Direction) String() string {
11     return [...]string{"North", "East", "South", "West"}[d]
12 }
13
14 func main() {
15     fmt.Println(South)
16 }
```

参考答案及解析：`South`。知识点：`iota` 的用法、类型的 `String()` 方法。

根据 `iota` 的用法推断出 `South` 的值是 2；另外，如果类型定义了 `String()` 方法，当使用 `fmt.Printf()`、`fmt.Print()` 和 `fmt.Println()` 会自动使用 `String()` 方法，实现字符串的打印。

Reference:

1. <https://wiki.jikexueyuan.com/project/the-way-to-go/10.7.html>
2. <https://www.sunansheng.com/archives/24.html>
3. <https://yourbasic.org/golang/iota/>

2.下面代码输出什么？

```

1  type Math struct {
2      x, y int
3  }
4
5  var m = map[string]Math{
6      "foo": Math{2, 3},
7  }
8
9  func main() {
10     m["foo"].x = 4
11     fmt.Println(m["foo"].x)
12 }

```

- A. 4
- B. compilation error

参考答案及解析：B，编译报错 `cannot assign to struct field m["foo"].x in map`。错误原因：对于类似 `x = y` 的赋值操作，必须知道 `x` 的地址，才能够将 `y` 的值赋给 `x`，但 go 中的 map 的 value 本身是不可寻址的。

有两个解决办法：

a.使用临时变量

```

1  type Math struct {
2      x, y int
3  }
4
5  var m = map[string]Math{
6      "foo": Math{2, 3},
7  }
8
9  func main() {
10     tmp := m["foo"]
11     tmp.x = 4
12     m["foo"] = tmp
13     fmt.Println(m["foo"].x)
14 }

```

b.修改数据结构

```

1  type Math struct {
2      x, y int
3  }
4
5  var m = map[string]*Math{
6      "foo": &Math{2, 3},
7  }
8
9  func main() {
10     m["foo"].x = 4
11     fmt.Println(m["foo"].x)
12     fmt.Printf("%#v", m["foo"])    // %#v 格式化输出详细信息
13 }

```

references:

0. https://blog.csdn.net/gq_36431213/article/details/82805043
1. <https://www.cnblogs.com/DillGao/p/7930674.html>
2. https://haobook.readthedocs.io/zh_CN/latest/periodical/201611/zhanghan.html
3. https://suraj.pro/post/golang_workaround/
4. <https://blog.ijun.org/2017/07/cannot-assign-to-struct-field-in-map.html>

第 28 天

1. 下面的代码有什么问题?

```

1  func main() {
2      fmt.Println([...]int{1} == [2]int{1})
3      fmt.Println([]int{1} == []int{1})
4  }

```

参考答案及解析：有两处错误

- go 中不同类型是不能比较的，而数组长度是数组类型的一部分，所以 `[...]int{1}` 和 `[2]int{1}` 是两种不同的类型，不能比较；
- 切片是不能比较的；

2. 下面这段代码输出什么?

```

1  var p *int
2
3  func foo() (*int, error) {
4      var i int = 5
5      return &i, nil
6  }
7
8  func bar() {
9      //use p
10     fmt.Println(*p)

```

```

11 }
12
13 func main() {
14     p, err := foo()
15     if err != nil {
16         fmt.Println(err)
17         return
18     }
19     bar()
20     fmt.Println(*p)
21 }

```

- A. 5 5
- B. runtime error

参考答案及解析：B。知识点：变量作用域。问题出在操作符 `:=`，对于使用 `:=` 定义的变量，如果新变量与同名已定义的变量不在同一个作用域中，那么 Go 会新定义这个变量。对于本例来说，`main()` 函数里的 `p` 是新定义的变量，会遮住全局变量 `p`，导致执行到 `bar()` 时程序，全局变量 `p` 依然还是 `nil`，程序随即 Crash。

正确的做法是将 `main()` 函数修改为：

```

1 func main() {
2     var err error
3     p, err = foo()
4     if err != nil {
5         fmt.Println(err)
6         return
7     }
8     bar()
9     fmt.Println(*p)
10 }

```

这道题目引自 Tony Bai 老师的一篇文章，原文讲的很详细，推荐。 <https://tonybai.com/2015/01/13/a-hole-about-variable-scope-in-golang/>

第 29 天

1.下面这段代码能否正常结束？

```

1 func main() {
2     v := []int{1, 2, 3}
3     for i := range v {
4         v = append(v, i)
5     }
6 }

```

参考答案及解析：不会出现死循环，能正常结束。

循环次数在循环开始前就已经确定，循环内改变切片的长度，不影响循环次数。

2.下面这段代码输出什么？为什么？

```
1 func main() {
2
3     var m = [...]int{1, 2, 3}
4
5     for i, v := range m {
6         go func() {
7             fmt.Println(i, v)
8         }()
9     }
10
11     time.Sleep(time.Second * 3)
12 }
```

参考答案及解析：

```
1 2 3
2 2 3
3 2 3
```

for range 使用短变量声明 (:=) 的形式迭代变量，需要注意的是，变量 i、v 在每次循环体中都会被重用，而不是重新声明。

各个 goroutine 中输出的 i、v 值都是 for range 循环结束后的 i、v 最终值，而不是各个 goroutine 启动时的 i、v 值。可以理解为闭包引用，使用的是上下文环境的值。两种可行的 fix 方法：

a.使用函数传递

```
1 for i, v := range m {
2     go func(i,v int) {
3         fmt.Println(i, v)
4     }(i,v)
5 }
```

b.使用临时变量保留当前值

```
1 for i, v := range m {
2     i := i          // 这里的 := 会重新声明变量，而不是重用
3     v := v
4     go func() {
5         fmt.Println(i, v)
6     }()
7 }
```

reference:

1.<https://tonybai.com/2015/09/17/7-things-you-may-not-pay-attention-to-in-go/>

第 30 天

1.下面这段代码输出什么?

```
1 func f(n int) (r int) {
2     defer func() {
3         r += n
4         recover()
5     }()
6
7     var f func()
8
9     defer f()
10    f = func() {
11        r += 2
12    }
13    return n + 1
14 }
15
16 func main() {
17     fmt.Println(f(3))
18 }
```

参考答案及解析：7。

2.下面这段代码输出什么?

```
1 func main() {
2     var a = [5]int{1, 2, 3, 4, 5}
3     var r [5]int
4
5     for i, v := range a {
6         if i == 0 {
7             a[1] = 12
8             a[2] = 13
9         }
10        r[i] = v
11    }
12    fmt.Println("r = ", r)
13    fmt.Println("a = ", a)
14 }
```

参考答案及解析：

```
1 r = [1 2 3 4 5]
2 a = [1 12 13 4 5]
```

range 表达式是副本参与循环，就是说例子中参与循环的是 a 的副本，而不是真正的 a。就这个例子来说，假设 b 是 a 的副本，则 range 循环代码是这样的：

```
1 for i, v := range b {
2     if i == 0 {
3         a[1] = 12
4         a[2] = 13
5     }
6     r[i] = v
7 }
```

因此无论 a 被如何修改，其副本 b 依旧保持原值，并且参与循环的是 b，因此 v 从 b 中取出的仍旧是 a 的原值，而非修改后的值。

如果想要 r 和 a 一样输出，修复办法：

```
1 func main() {
2     var a = [5]int{1, 2, 3, 4, 5}
3     var r [5]int
4
5     for i, v := range &a {
6         if i == 0 {
7             a[1] = 12
8             a[2] = 13
9         }
10        r[i] = v
11    }
12    fmt.Println("r = ", r)
13    fmt.Println("a = ", a)
14 }
```

输出：

```
1 r = [1 12 13 4 5]
2 a = [1 12 13 4 5]
```

修复代码中，使用 `*[5]int` 作为 range 表达式，其副本依旧是一个指向原数组 a 的指针，因此后续所有循环中均是 &a 指向的原数组亲自参与的，因此 v 能从 &a 指向的原数组中取出 a 修改后的值。

reference: <https://tonybai.com/2015/09/17/7-things-you-may-not-pay-attention-to-in-go/>

第 31 天

1.下面这段代码输出什么？

```

1 func change(s ...int) {
2     s = append(s,3)
3 }
4
5 func main() {
6     slice := make([]int,5,5)
7     slice[0] = 1
8     slice[1] = 2
9     change(slice...)
10    fmt.Println(slice)
11    change(slice[0:2]...)
12    fmt.Println(slice)
13 }

```

参考答案及解析：

```

1 [1 2 0 0 0]
2 [1 2 3 0 0]

```

知识点：可变函数、append()操作。Go 提供的语法糖 `...`，可以将 slice 传进可变函数，不会创建新的切片。第一次调用 change() 时，append() 操作使切片底层数组发生了扩容，原 slice 的底层数组不会改变；第二次调用 change() 函数时，使用了操作符 `[i,j]` 获得一个新的切片，假定为 slice1，它的底层数组和原切片底层数组是重合的，不过 slice1 的长度、容量分别是 2、5，所以在 change() 函数中对 slice1 底层数组的修改会影响到原切片。

2.下面这段代码输出什么？

```

1 func main() {
2     var a = []int{1, 2, 3, 4, 5}
3     var r [5]int
4
5     for i, v := range a {
6         if i == 0 {
7             a[1] = 12
8             a[2] = 13
9         }
10        r[i] = v
11    }
12    fmt.Println("r = ", r)
13    fmt.Println("a = ", a)
14 }

```

参考答案及解析：

```

1 r = [1 12 13 4 5]
2 a = [1 12 13 4 5]

```

这道题是昨天第二题的一个解决办法，这的 a 是一个切片，那切片是怎么实现的呢？切片在 go 的内部结构有一个指向底层数组的指针，当 range 表达式发生复制时，副本的指针依旧指向原底层数组，所以对切片的修改都会反应到底层数组上，所以通过 v 可以获得修改后的数组元素。

引自：<https://tonybai.com/2015/09/17/7-things-you-may-not-pay-attention-to-in-go/>

第 32 天

1.下面这段代码输出结果正确吗？

```
1 type Foo struct {
2     bar string
3 }
4 func main() {
5     s1 := []Foo{
6         {"A"},
7         {"B"},
8         {"C"},
9     }
10    s2 := make([]*Foo, len(s1))
11    for i, value := range s1 {
12        s2[i] = &value
13    }
14    fmt.Println(s1[0], s1[1], s1[2])
15    fmt.Println(s2[0], s2[1], s2[2])
16 }
```

```
1 输出：
2 {A} {B} {C}
3 &{A} &{B} &{C}
```

参考答案及解析：s2 的输出结果错误。s2 的输出是 `&{C} &{C} &{C}`，在 29 天我们提到过，for range 使用短变量声明(=)的形式迭代变量时，变量 i、value 在每次循环体中都会被重用，而不是重新声明。所以 s2 每次填充的都是临时变量 value 的地址，而在最后一次循环中，value 被赋值为{c}。因此，s2 输出的时候显示出了三个 &{c}。

可行的解决办法如下：

```
1 for i := range s1 {
2     s2[i] = &s1[i]
3 }
```

2.下面代码里的 counter 的输出值？

```
1 func main() {
2
```

```

3   var m = map[string]int{
4       "A": 21,
5       "B": 22,
6       "C": 23,
7   }
8   counter := 0
9   for k, v := range m {
10      if counter == 0 {
11          delete(m, "A")
12      }
13      counter++
14      fmt.Println(k, v)
15  }
16  fmt.Println("counter is ", counter)
17  }

```

- A. 2
- B. 3
- C. 2 或 3

参考答案及解析：C。for range map 是无序的，如果第一次循环到 A，则输出 3；否则输出 2。

第 33 天

1.关于协程，下面说法正确是（）

- A. 协程和线程都可以实现程序的并发执行；
- B. 线程比协程更轻量级；
- C. 协程不存在死锁问题；
- D. 通过 channel 来进行协程间的通信；

参考答案及解析：AD。

2.关于循环语句，下面说法正确的有（）

- A. 循环语句既支持 for 关键字，也支持 while 和 do-while；
- B. 关键字 for 的基本使用方法与 C/C++ 中没有任何差异；
- C. for 循环支持 continue 和 break 来控制循环，但是它提供了一个更高级的 break，可以选择中断哪一个循环；
- D. for 循环不支持以逗号为间隔的多个赋值语句，必须使用平行赋值的方式来初始化多个变量；

参考答案及解析：CD。

3.下面代码输出正确的是？

```
1 func main() {
2     i := 1
3     s := []string{"A", "B", "C"}
4     i, s[i-1] = 2, "Z"
5     fmt.Printf("s: %v \n", s)
6 }
```

- A. s: [Z,B,C]
- B. s: [A,Z,C]

参考答案及解析：A。知识点：多重赋值。

多重赋值分为两个步骤，有先后顺序：

- 计算等号左边的索引表达式和取址表达式，接着计算等号右边的表达式；
- 赋值；

所以本例，会先计算 `s[i-1]`，等号右边是两个表达式是常量，所以赋值运算等同于 `i, s[0] = 2, "Z"`。

第 34 天

1.关于类型转化，下面选项正确的是？

```
1 A.
2 type MyInt int
3 var i int = 1
4 var j MyInt = i
5
6 B.
7 type MyInt int
8 var i int = 1
9 var j MyInt = (MyInt)i
10
11 C.
12 type MyInt int
13 var i int = 1
14 var j MyInt = MyInt(i)
15
16 D.
17 type MyInt int
18 var i int = 1
19 var j MyInt = i.(MyInt)
```

参考答案及解析：C。知识点：强制类型转化。

2.关于switch语句，下面说法正确的有？

- A. 条件表达式必须为常量或者整数；
- B. 单个case中，可以出现多个结果选项；
- C. 需要用break来明确退出一个case；

- D. 只有在case中明确添加fallthrough关键字, 才会继续执行紧跟的下一个case;

参考答案及解析: BD。

3.如果 Add() 函数的调用代码为:

```
1 func main() {
2     var a Integer = 1
3     var b Integer = 2
4     var i interface{} = &a
5     sum := i.(*Integer).Add(b)
6     fmt.Println(sum)
7 }
```

则Add函数定义正确的是()

```
1 A.
2 type Integer int
3 func (a Integer) Add(b Integer) Integer {
4     return a + b
5 }
6
7 B.
8 type Integer int
9 func (a Integer) Add(b *Integer) Integer {
10    return a + *b
11 }
12
13 C.
14 type Integer int
15 func (a *Integer) Add(b Integer) Integer {
16    return *a + b
17 }
18
19 D.
20 type Integer int
21 func (a *Integer) Add(b *Integer) Integer {
22    return *a + *b
23 }
```

参考答案及解析: AC。知识点: 类型断言、方法集。

第 35 天

1.关于 bool 变量 b 的赋值, 下面错误的用法是?

- A. b = true
- B. b = 1
- C. b = bool(1)

- D. b = (1 == 2)

参考答案及解析：BC。

2.关于变量的自增和自减操作，下面语句正确的是？

```
1  A.
2  i := 1
3  i++
4
5  B.
6  i := 1
7  j = i++
8
9  C.
10 i := 1
11 ++i
12
13 D.
14 i := 1
15 i--
```

参考答案及解析：AD。知识点：自增自减操作。i++ 和 i-- 在 Go 语言中是语句，不是表达式，因此不能赋值给另外的变量。此外没有 ++i 和 --i。

3.关于GetPodAction定义，下面赋值正确的是

```
1  type Fragment interface {
2      Exec(transInfo *TransInfo) error
3  }
4  type GetPodAction struct {
5  }
6  func (g GetPodAction) Exec(transInfo *TransInfo) error {
7      ...
8      return nil
9  }
```

- A. var fragment Fragment = new(GetPodAction)
- B. var fragment Fragment = GetPodAction
- C. var fragment Fragment = &GetPodAction{}
- D. var fragment Fragment = GetPodAction{}

参考答案及解析：ACD。

第 36 天

1.关于函数声明，下面语法正确的是？

- A. func f(a, b int) (value int, err error)
- B. func f(a int, b int) (value int, err error)
- C. func f(a, b int) (value int, error)
- D. func f(a int, b int) (int, int, error)

参考答案及解析：ABD。

2.关于整型切片的初始化，下面正确的是？

- A. s := make([]int)
- B. s := make([]int, 0)
- C. s := make([]int, 5, 10)
- D. s := []int{1, 2, 3, 4, 5}

参考答案及解析：BCD。

3.下面代码会触发异常吗？请说明。

```
1 func main() {
2     runtime.GOMAXPROCS(1)
3     int_chan := make(chan int, 1)
4     string_chan := make(chan string, 1)
5     int_chan <- 1
6     string_chan <- "hello"
7     select {
8     case value := <-int_chan:
9         fmt.Println(value)
10    case value := <-string_chan:
11        panic(value)
12    }
13 }
```

参考答案及解析：`select` 会随机选择一个可用通道做收发操作，所以可能触发异常，也可能不会。

第 37 天

1.关于channel的特性，下面说法正确的是

- A. 给一个 nil channel 发送数据，造成永远阻塞
- B. 从一个 nil channel 接收数据，造成永远阻塞
- C. 给一个已经关闭的 channel 发送数据，引起 panic
- D. 从一个已经关闭的 channel 接收数据，如果缓冲区中为空，则返回一个零值

参考答案及解析：ABCD。

2.下面代码有什么问题？

```

1  const i = 100
2  var j = 123
3
4  func main() {
5      fmt.Println(&j, j)
6      fmt.Println(&i, i)
7  }

```

参考答案及解析：编译报错 `cannot take the address of i`。知识点：常量。常量不同于变量的在运行期分配内存，常量通常会被编译器在预处理阶段直接展开，作为指令数据使用，所以常量无法寻址。

3.下面代码能否编译通过？如果通过，输出什么？

```

1  func GetValue(m map[int]string, id int) (string, bool) {
2
3      if _, exist := m[id]; exist {
4          return "exist", true
5      }
6      return nil, false
7  }
8  func main() {
9      intmap := map[int]string{
10         1: "a",
11         2: "b",
12         3: "c",
13     }
14
15     v, err := GetValue(intmap, 3)
16     fmt.Println(v, err)
17 }

```

参考答案及解析：不能通过编译。知识点：函数返回值类型。nil 可以用作 interface、function、pointer、map、slice 和 channel 的“空值”。但是如果不特别指定的话，Go 语言不能识别类型，所以会报错：`cannot use nil as type string in return argument`。

第 38 天

1.关于异常的触发，下面说法正确的是？

- A. 空指针解析；
- B. 下标越界；
- C. 除数为0；
- D. 调用panic函数；

参考答案及解析：ABCD。

2.下面代码输出什么？

```

1 func main() {
2     x := []string{"a", "b", "c"}
3     for v := range x {
4         fmt.Print(v)
5     }
6 }

```

参考答案及解析：012。注意区别下面代码段：

```

1 func main() {
2     x := []string{"a", "b", "c"}
3     for _, v := range x {
4         fmt.Print(v)    //输出 abc
5     }
6 }

```

3.下面这段代码能否编译通过？如果通过，输出什么？

```

1 type User struct{}
2 type User1 User
3 type User2 = User
4
5 func (i User1) m1() {
6     fmt.Println("m1")
7 }
8 func (i User) m2() {
9     fmt.Println("m2")
10 }
11
12 func main() {
13     var i1 User1
14     var i2 User2
15     i1.m1()
16     i2.m2()
17 }

```

参考答案及解析：能，输出 `m1 m2`，第 2 行代码基于类型 `User` 创建了新类型 `User1`，第 3 行代码是创建了 `User` 的类型别名 `User2`，注意使用 `=` 定义类型别名。因为 `User2` 是别名，完全等价于 `User`，所以 `User2` 具有 `User` 所有的方法。但是 `i1.m2()` 是不能执行的，因为 `User1` 没有定义该方法。

第 39 天

1.关于无缓冲和有冲突的channel，下面说法正确的是？

- A. 无缓冲的channel是默认的缓冲为1的channel；
- B. 无缓冲的channel和有缓冲的channel都是同步的；
- C. 无缓冲的channel和有缓冲的channel都是非同步的；
- D. 无缓冲的channel是同步的，而有缓冲的channel是非同步的；

参考答案及解析：D。

2.下面代码是否能编译通过？如果通过，输出什么？

```
1 func Foo(x interface{}) {
2     if x == nil {
3         fmt.Println("empty interface")
4         return
5     }
6     fmt.Println("non-empty interface")
7 }
8 func main() {
9     var x *int = nil
10    Foo(x)
11 }
```

参考答案及解析：`non-empty interface` 考点：interface 的内部结构，我们知道接口除了有静态类型，还有动态类型和动态值，当且仅当动态值和动态类型都为 nil 时，接口类型值才为 nil。这里的 x 的动态类型是 `*int`，所以 x 不为 nil。

3.下面代码输出什么？

```
1 func main() {
2     ch := make(chan int, 100)
3     // A
4     go func() {
5         for i := 0; i < 10; i++ {
6             ch <- i
7         }
8     }()
9     // B
10    go func() {
11        for {
12            a, ok := <-ch
13            if !ok {
14                fmt.Println("close")
15                return
16            }
17            fmt.Println("a: ", a)
18        }
19    }()
20    close(ch)
21    fmt.Println("ok")
22    time.Sleep(time.Second * 10)
23 }
```

参考答案及解析：程序抛异常。先定义下，第一个协程为 A 协程，第二个协程为 B 协程；当 A 协程还没起时，主协程已经将 channel 关闭了，当 A 协程往关闭的 channel 发送数据时会 panic，panic: send on closed channel。

第 40 天

1.关于select机制，下面说法正确的是？

- A. select机制用来处理异步IO问题；
- B. select机制最大的一条限制就是每个case语句里必须是一个IO操作；
- C. go语言在语言级别支持select关键字；
- D. select关键字的用法与switch语句非常类似，后面要带判断条件；

参考答案及解析：ABC。

2.下面的代码有什么问题？

```
1 func Stop(stop <-chan bool) {
2     close(stop)
3 }
```

参考答案及解析：有方向的 channel 不可被关闭。

3.下面这段代码存在什么问题？

```
1 type Param map[string]interface{}
2
3 type Show struct {
4     *Param
5 }
6
7 func main() {
8     s := new(Show)
9     s.Param["day"] = 2
10 }
```

参考答案及解析：存在两个问题：1.map 需要初始化才能使用；2.指针不支持索引。修复代码如下：

```
1 func main() {
2     s := new(Show)
3     // 修复代码
4     p := make(Param)
5     p["day"] = 2
6     s.Param = &p
7     tmp := *s.Param
8     fmt.Println(tmp["day"])
9 }
```

第 41 天

1.下面代码编译能通过吗?

```
1 func main()
2 {
3     fmt.Println("hello world")
4 }
```

参考答案及解析：编译错误。

```
1 syntax error: unexpected semicolon or newline before {
```

Go 语言中，大括号不能放在单独的一行。

正确的代码如下：

```
1 func main() {
2     fmt.Println("works")
3 }
```

引自：<http://devs.cloudimmunity.com/gotchas-and-common-mistakes-in-go-golang/>

2.下面这段代码输出什么?

```
1 var x = []int{2: 2, 3, 0: 1}
2
3 func main() {
4     fmt.Println(x)
5 }
```

参考答案及解析：输出 `[1 0 2 3]`，字面量初始化切片时候，可以指定索引，没有指定索引的元素会在前一个索引基础之上加一，所以输出 `[1 0 2 3]`，而不是 `[1 3 2]`。

3.下面这段代码输出什么?

```
1 func incr(p *int) int {
2     *p++
3     return *p
4 }
5 func main() {
6     v := 1
7     incr(&v)
8     fmt.Println(v)
9 }
```

参考答案及解析：2。知识点：指针。p 是指针变量，指向变量 v，*p++ 操作的意思是取出变量 v 的值并执行加一操作，所以 v 的最终值是 2。

第 42 天

1.请指出下面代码的错误？

```
1 package main
2
3 var gvar int
4
5 func main() {
6     var one int
7     two := 2
8     var three int
9     three = 3
10
11     func(unused string) {
12         fmt.Println("Unused arg. No compile error")
13     }("what?")
14 }
```

参考答案及解析：变量 one、two 和 three 声明未使用。知识点：未使用变量。如果有未使用的变量代码将编译失败。但也有例外，函数中声明的变量必须要使用，但可以有未使用的全局变量。函数的参数未使用也是可以的。

如果你给未使用的变量分配了一个新值，代码也还是会编译失败。你需要在某个地方使用这个变量，才能让编译器愉快的编译。

修复代码：

```
1 func main() {
2     var one int
3     _ = one
4
5     two := 2
6     fmt.Println(two)
7
8     var three int
9     three = 3
10    one = three
11
12    var four int
13    four = four
14 }
```

另一个选择是注释掉或者移除未使用的变量。

引自：<http://devs.cloudimmunity.com/gotchas-and-common-mistakes-in-go-golang/>

2.下面代码输出什么？

```
1 type ConfigOne struct {
2     Daemon string
3 }
4
5 func (c *ConfigOne) String() string {
6     return fmt.Sprintf("print: %v", c)
7 }
8
9 func main() {
10    c := &ConfigOne{}
11    c.String()
12 }
```

参考答案及解析：运行时错误。如果类型实现 String() 方法，当格式化输出时会自动使用 String() 方法。上面这段代码是在该类型的 String() 方法内使用格式化输出，导致递归调用，最后抛错。

```
1 runtime: goroutine stack exceeds 1000000000-byte limit
2 fatal error: stack overflow
```

3.下面代码输出什么？

```
1 func main() {
2     var a = []int{1, 2, 3, 4, 5}
3     var r = make([]int, 0)
4
5     for i, v := range a {
6         if i == 0 {
7             a = append(a, 6, 7)
8         }
9
10        r = append(r, v)
11    }
12
13    fmt.Println(r)
14 }
```

参考答案及解析：`[1 2 3 4 5]`。a 在 for range 过程中增加了两个元素，len 由 5 增加到 7，但 for range 时会使用 a 的副本 a' 参与循环，副本的 len 依旧是 5，因此 for range 只会循环 5 次，也就只获取 a 对应的底层数组的前 5 个元素。

第 43 天

1.下面的代码有什么问题？

```
1 import (
2     "fmt"
3     "log"
4     "time"
5 )
6 func main() {
7 }
```

参考答案及解析：导入的包没有被使用。如果引入一个包，但是未使用其中任何函数、接口、结构体或变量的话，代码将编译失败。

如果你真的需要引入包，可以使用下划线操作符，`_`，来作为这个包的名字，从而避免失败。下划线操作符用于引入，但不使用。

我们还可以注释或者移除未使用的包。

修复代码：

```
1 import (
2     _ "fmt"
3     "log"
4     "time"
5 )
6 var _ = log.Println
7 func main() {
8     _ = time.Now
9 }
```

引自：<http://devs.cloudimmunity.com/gotchas-and-common-mistakes-in-go-golang/>

2.下面代码输出什么？

```
1 func main() {
2     x := interface{}(nil)
3     y := (*int)(nil)
4     a := y == x
5     b := y == nil
6     _, c := x.(interface{})
7     println(a, b, c)
8 }
```

- A. true true false
- B. false true true
- C. true true true
- D. false true false

参考答案及解析：D。知识点：类型断言。类型断言语法：i.(Type)，其中 i 是接口，Type 是类型或接口。编译时会自动检测 i 的动态类型与 Type 是否一致。但是，如果动态类型不存在，则断言总是失败。参考文章《接口2》

3.下面代码有几处错误的地方？请说明原因。

```
1 func main() {
2
3     var s []int
4     s = append(s,1)
5
6     var m map[string]int
7     m["one"] = 1
8 }
```

参考答案及解析：有 1 处错误，不能对 nil 的 map 直接赋值，需要使用 make() 初始化。但可以使用 append() 函数对为 nil 的 slice 增加元素。

修复代码：

```
1 func main() {
2     var m map[string]int
3     m = make(map[string]int)
4     m["one"] = 1
5 }
```

第 44 天

1.下面代码有什么问题？

```
1 func main() {
2     m := make(map[string]int,2)
3     cap(m)
4 }
```

参考答案及解析：问题：使用 cap() 获取 map 的容量。1.使用 make 创建 map 变量时可以指定第二个参数，不过会被忽略。2.cap() 函数适用于数组、数组指针、slice 和 channel，不适用于 map，可以使用 len() 返回 map 的元素个数。

引自：<http://devs.cloudimmunity.com/gotchas-and-common-mistakes-in-go-golang/index.html>

2.下面的代码有什么问题？

```
1 func main() {
2     var x = nil
3     _ = x
4 }
```

参考答案及解析：nil 用于表示 interface、函数、maps、slices 和 channels 的“零值”。如果不指定变量的类型，编译器猜不出变量的具体类型，导致编译错误。

修复代码：

```
1 func main() {
2     var x interface{} = nil
3     _ = x
4 }
```

3.下面代码能编译通过吗？

```
1 type info struct {
2     result int
3 }
4
5 func work() (int,error) {
6     return 13,nil
7 }
8
9 func main() {
10    var data info
11
12    data.result, err := work()
13    fmt.Printf("info: %+v\n",data)
14 }
```

参考答案及解析：编译失败。

```
1 non-name data.result on left side of :=
```

不能使用短变量声明设置结构体字段值，修复代码：

```
1 func main() {
2     var data info
3
4     var err error
5     data.result, err = work() //ok
6     if err != nil {
7         fmt.Println(err)
8         return
9     }
10
11    fmt.Println(data)
12 }
```

引自：<http://devs.cloudimmunity.com/gotchas-and-common-mistakes-in-go-golang/index.html>

第 45 天

1.下面代码有什么错误?

```
1 func main() {
2     one := 0
3     one := 1
4 }
```

参考答案及解析：变量重复声明。不能在单独的声明中重复声明一个变量，但在多变量声明的时候是可以的，但必须保证至少有一个变量是新声明的。

修复代码：

```
1 func main() {
2     one := 0
3     one, two := 1, 2
4     one, two = two, one
5 }
```

引自：<http://devs.cloudimmunity.com/gotchas-and-common-mistakes-in-go-golang/>

2.下面代码有什么问题?

```
1 func main() {
2     x := []int{
3         1,
4         2
5     }
6     _ = x
7 }
```

参考答案及解析：编译错误，第四行代码没有逗号。用字面量初始化数组、slice 和 map 时，最好是在每个元素后面加上逗号，即使是声明在一行或者多行都不会出错。

修复代码：

```
1 func main() {
2     x := []int{ // 多行
3         1,
4         2,
5     }
6     x = x
7
8     y := []int{3,4,} // 一行 no error
9     y = y
10 }
```

引自: <http://devs.cloudimmunity.com/gotchas-and-common-mistakes-in-go-golang/index.html>

3.下面代码输出什么?

```
1 func test(x byte) {
2     fmt.Println(x)
3 }
4
5 func main() {
6     var a byte = 0x11
7     var b uint8 = a
8     var c uint8 = a + b
9     test(c)
10 }
```

参考答案及解析: 34。与 rune 是 int32 的别名一样, byte 是 uint8 的别名, 别名类型无序转换, 可直接转换。

第 46 天

1.下面的代码有什么问题?

```
1 func main() {
2     const x = 123
3     const y = 1.23
4     fmt.Println(x)
5 }
```

参考答案及解析: 编译可以通过。知识点: 常量。常量是一个简单值的标识符, 在程序运行时, 不会被修改的量。不像变量, 常量未使用是能编译通过的。

2.下面代码输出什么?

```
1 const (
2     x uint16 = 120
3     y
4     s = "abc"
5     z
6 )
7
8 func main() {
9     fmt.Printf("%T %v\n", y, y)
10    fmt.Printf("%T %v\n", z, z)
11 }
```

参考答案及解析: 知识点: 常量。

输出:

```
1 uint16 120
2 string abc
```

常量组中如不指定类型和初始化值，则与上一行非空常量右值相同

3.下面代码有什么问题？

```
1 func main() {
2     var x string = nil
3
4     if x == nil {
5         x = "default"
6     }
7 }
```

参考答案及解析：将 nil 分配给 string 类型的变量。这是个大多数新手会犯的错误。修复代码：

```
1 func main() {
2     var x string //defaults to "" (zero value)
3
4     if x == "" {
5         x = "default"
6     }
7 }
```

引自：<http://devs.cloudimmunity.com/gotchas-and-common-mistakes-in-go-golang/index.html>

第 47 天

1.下面的代码有什么问题？

```
1 func main() {
2     data := []int{1,2,3}
3     i := 0
4     ++i
5     fmt.Println(data[i++])
6 }
```

参考答案及解析：对于自增、自减，需要注意：

- 自增、自减不在是运算符，只能作为独立语句，而不是表达式；
- 不像其他语言，Go 语言中不支持 ++i 和 --i 操作；

表达式通常是求值代码，可作为右值或参数使用。而语句表示完成一个任务，比如 if、for 语句等。表达式可作为语句使用，但语句不能当做表达式。

修复代码：

```

1 func main() {
2     data := []int{1,2,3}
3     i := 0
4     i++
5     fmt.Println(data[i])
6 }

```

2.下面代码最后一行输出什么？请说明原因。

```

1 func main() {
2     x := 1
3     fmt.Println(x)
4     {
5         fmt.Println(x)
6         i,x := 2,2
7         fmt.Println(i,x)
8     }
9     fmt.Println(x) // print ?
10 }

```

参考答案及解析：输出 1。知识点：变量隐藏。使用变量简短声明符号 := 时，如果符号左边有多个变量，只需要保证至少有一个变量是新声明的，并对已定义的变量尽进行赋值操作。但如果出现作用域之后，就会导致变量隐藏的问题，就像这个例子一样。

这个坑很容易挖，但又很难发现。即使对于经验丰富的 Go 开发者而言，这也是一个非常常见的陷阱。

引自：<http://devs.cloudimmunity.com/gotchas-and-common-mistakes-in-go-golang/>

第 48 天

1.下面代码有什么问题？

```

1 type foo struct {
2     bar int
3 }
4
5 func main() {
6     var f foo
7     f.bar, tmp := 1, 2
8 }

```

参考答案及解析：编译错误：

```

1 non-name f.bar on left side of :=

```

:= 操作符不能用于结构体字段赋值。

2.下面的代码输出什么？

```
1 func main() {
2     fmt.Println(~2)
3 }
```

参考答案及解析：编译错误。

```
1 invalid character U+007E '~'
```

很多语言都是采用 ~ 作为按位取反运算符，Go 里面采用的是 ^。按位取反之后返回一个每个 bit 位都取反的数，对于有符号的整数来说，是按照补码进行取反操作的（快速计算方法：对数 a 取反，结果为 $-(a+1)$ ），对于无符号整数来说就是按位取反。例如：

```
1 func main() {
2     var a int8 = 3
3     var b uint8 = 3
4     var c int8 = -3
5
6     fmt.Printf("^%b=%b %d\n", a, ^a, ^a) // ^11=-100 -4
7     fmt.Printf("^%b=%b %d\n", b, ^b, ^b) // ^11=11111100 252
8     fmt.Printf("^%b=%b %d\n", c, ^c, ^c) // ^-11=10 2
9 }
```

另外需要注意的是，如果作为二元运算符，^ 表示按位异或，即：对应位相同为 0，相异为 1。例如：

```
1 func main() {
2     var a int8 = 3
3     var c int8 = 5
4
5     fmt.Printf("a: %08b\n",a)
6     fmt.Printf("c: %08b\n",c)
7     fmt.Printf("a^c: %08b\n",a ^ c)
8 }
```

给大家重点介绍下这个操作符 &^，按位置零，例如： $z = x \&^ y$ ，表示如果 y 中的 bit 位为 1，则 z 对应 bit 位为 0，否则 z 对应 bit 位等于 x 中相应的 bit 位的值。

不知道大家发现没有，我们还可以这样理解或操作符 |，表达式 $z = x | y$ ，如果 y 中的 bit 位为 1，则 z 对应 bit 位为 1，否则 z 对应 bit 位等于 x 中相应的 bit 位的值，与 &^ 完全相反。

```
1 var x uint8 = 214
2 var y uint8 = 92
3 fmt.Printf("x: %08b\n",x)
4 fmt.Printf("y: %08b\n",y)
5 fmt.Printf("x | y: %08b\n",x | y)
6 fmt.Printf("x &^ y: %08b\n",x &^ y)
```

输出：

```
1 x: 11010110
2 y: 01011100
3 x | y: 11011110
4 x &^ y: 10000010
```

第 49 天

1.下面代码输出什么?

```
1 func main() {
2     var ch chan int
3     select {
4     case v, ok := <-ch:
5         println(v, ok)
6     default:
7         println("default")
8     }
9 }
```

参考答案及解析: default。ch 为 nil, 读写都会阻塞。

2.下面这段代码输出什么?

```
1 type People struct {
2     name string `json:"name"`
3 }
4
5 func main() {
6     js := `{
7         "name":"seekload"
8     }`
9     var p People
10    err := json.Unmarshal([]byte(js), &p)
11    if err != nil {
12        fmt.Println("err: ", err)
13        return
14    }
15    fmt.Println(p)
16 }
```

参考答案及解析: 输出 {}。知识点: 结构体访问控制, 因为 name 首字母是小写, 导致其他包不能访问, 所以输出为空结构体。修复代码:

```
1 type People struct {
2     Name string `json:"name"`
3 }
```

第 50 天

1.下面这段代码输出什么?

```
1 type T struct {
2     ls []int
3 }
4
5 func foo(t T) {
6     t.ls[0] = 100
7 }
8
9 func main() {
10    var t = T{
11        ls: []int{1, 2, 3},
12    }
13
14    foo(t)
15    fmt.Println(t.ls[0])
16 }
```

- A. 1
- B. 100
- C. compilation error

参考答案及解析: B。调用 foo() 函数时虽然是传值, 但 foo() 函数中, 字段 ls 依旧可以看成是指向底层数组的指针。

2.下面代码输出什么?

```
1 func main() {
2     isMatch := func(i int) bool {
3         switch(i) {
4             case 1:
5             case 2:
6                 return true
7         }
8         return false
9     }
10
11    fmt.Println(isMatch(1))
12    fmt.Println(isMatch(2))
13 }
```

参考答案及解析: false true。Go 语言的 switch 语句虽然没有"break", 但如果 case 完成程序会默认 break, 可以在 case 语句后面加上关键字 fallthrough, 这样就会接着走下一个 case 语句 (不用匹配后续条件表达式)。或者, 利用 case 可以匹配多个值的特性。

修复代码:

```
1 func main() {
2     isMatch := func(i int) bool {
3         switch(i) {
4             case 1:
5                 fallthrough
6             case 2:
7                 return true
8         }
9         return false
10    }
11
12    fmt.Println(isMatch(1))    // true
13    fmt.Println(isMatch(2))    // true
14
15    match := func(i int) bool {
16        switch(i) {
17            case 1,2:
18                return true
19        }
20        return false
21    }
22
23    fmt.Println(match(1))      // true
24    fmt.Println(match(2))      // true
25 }
```

第 51 天

1. 下面的代码能否正确输出?

```
1 func main() {
2     var fn1 = func() {}
3     var fn2 = func() {}
4
5     if fn1 != fn2 {
6         println("fn1 not equal fn2")
7     }
8 }
```

参考答案及解析: 编译错误

```
1 invalid operation: fn1 != fn2 (func can only be compared to nil)
```

函数只能与 nil 比较。

2. 下面代码输出什么?

```
1 type T struct {
2     n int
3 }
4
5 func main() {
6     m := make(map[int]T)
7     m[0].n = 1
8     fmt.Println(m[0].n)
9 }
```

- A. 1
- B. compilation error

参考答案及解析：B。编译错误：

```
1 cannot assign to struct field m[0].n in map
```

map[key]struct 中 struct 是不可寻址的，所以无法直接赋值。

修复代码：

```
1 type T struct {
2     n int
3 }
4
5 func main() {
6     m := make(map[int]T)
7
8     t := T{1}
9     m[0] = t
10    fmt.Println(m[0].n)
11 }
```

第 52 天

1. 下面的代码有什么问题？

```

1  type X struct {}
2
3  func (x *X) test() {
4      println(x)
5  }
6
7  func main() {
8
9      var a *X
10     a.test()
11
12     X{}.test()
13 }

```

参考答案及解析：X{} 是不可寻址的，不能直接调用方法。知识点：在方法中，指针类型的接收者必须是合法指针（包括 nil），或能获取实例地址。

修复代码：

```

1  func main() {
2
3      var a *X
4      a.test()    // 相当于 test(nil)
5
6      var x = X{}
7      x.test()
8  }

```

引自：《Go语言学习笔记》·方法

2.下面代码有什么不规范的地方吗？

```

1  func main() {
2      x := map[string]string{"one": "a", "two": "", "three": "c"}
3
4      if v := x["two"]; v == "" {
5          fmt.Println("no entry")
6      }
7  }

```

参考答案及解析：检查 map 是否含有某一元素，直接判断元素的值并不是一种合适的方式。最可靠的操作是使用访问 map 时返回的第二个值。

修复代码如下：

```

1 func main() {
2     x := map[string]string{"one": "a", "two": "", "three": "c"}
3
4     if _, ok := x["two"]; !ok {
5         fmt.Println("no entry")
6     }
7 }

```

引自: <http://devs.cloudimmunity.com/gotchas-and-common-mistakes-in-go-golang/index.html>

第 53 天

1.关于 channel 下面描述正确的是?

- A. 向已关闭的通道发送数据会引发 panic;
- B. 从已关闭的缓冲通道接收数据, 返回已缓冲数据或者零值;
- C. 无论接收还是接收, nil 通道都会阻塞;

参考答案及解析: ABC。

2.下面的代码有几处问题? 请详细说明。

```

1 type T struct {
2     n int
3 }
4
5 func (t *T) Set(n int) {
6     t.n = n
7 }
8
9 func getT() T {
10    return T{}
11 }
12
13 func main() {
14    getT().Set(1)
15 }

```

参考答案及解析: 有两处问题:

- 1.直接返回的 T{} 不可寻址;
- 2.不可寻址的结构体不能调用带结构体指针接收者的方法;

修复代码:

```

1 type T struct {
2     n int
3 }
4
5 func (t *T) Set(n int) {

```

```

6     t.n = n
7 }
8
9 func getT() T {
10     return T{}
11 }
12
13 func main() {
14     t := getT()
15     t.Set(2)
16     fmt.Println(t.n)
17 }

```

第 54 天

1. 下面的代码有什么问题?

```

1 func (n N) value(){
2     n++
3     fmt.Printf("v:%p,%v\n", &n, n)
4 }
5
6 func (n *N) pointer(){
7     *n++
8     fmt.Printf("v:%p,%v\n", n, *n)
9 }
10
11
12 func main() {
13
14     var a N = 25
15
16     p := &a
17     p1 := &p
18
19     p1.value()
20     p1.pointer()
21 }

```

参考答案及解析：编译错误：

```

1 calling method value with receiver p1 (type **N) requires explicit dereference
2 calling method pointer with receiver p1 (type **N) requires explicit dereference

```

不能使用多级指针调用方法。

2. 下面的代码输出什么?

```

1  type N int
2
3  func (n N) test(){
4      fmt.Println(n)
5  }
6
7  func main() {
8      var n N = 10
9      fmt.Println(n)
10
11     n++
12     f1 := N.test
13     f1(n)
14
15     n++
16     f2 := (*N).test
17     f2(&n)
18 }

```

参考答案及解析：10 11 12。知识点：方法表达式。通过类型引用的方法表达式会被还原成普通函数样式，接收者是第一个参数，调用时显示传参。类型可以是 T 或 *T，只要目标方法存在于该类型的方法集中就可以。

还可以直接使用方法表达式调用：

```

1  func main() {
2      var n N = 10
3
4      fmt.Println(n)
5
6      n++
7      N.test(n)
8
9      n++
10     (*N).test(&n)
11 }

```

引自：《Go语言学习笔记》·方法

第 55 天

1.关于 channel 下面描述正确的是？

- A. close() 可以用于只接收通道；
- B. 单向通道可以转换为双向通道；
- C. 不能在单向通道上做逆向操作（例如：只发送通道用于接收）；

参考答案及解析：C。

2.下面的代码有什么问题？

```
1 type T struct {
2     n int
3 }
4
5 func getT() T {
6     return T{}
7 }
8
9 func main() {
10    getT().n = 1
11 }
```

参考答案及解析：编译错误：

```
1 cannot assign to getT().n
```

直接返回的 T{} 无法寻址，不可直接赋值。

修复代码：

```
1 type T struct {
2     n int
3 }
4
5 func getT() T {
6     return T{}
7 }
8
9 func main() {
10    t := getT()
11    p := &t.n    // <=> p = &(t.n)
12    *p = 1
13    fmt.Println(t.n)
14 }
```

第 56 天

1. 下面的代码有什么问题？

```

1 package main
2
3 import "fmt"
4
5 func main() {
6     s := make([]int, 3, 9)
7     fmt.Println(len(s))
8     s2 := s[4:8]
9     fmt.Println(len(s2))
10 }

```

参考答案及解析：代码没问题，输出 3 4。从一个基础切片派生出的子切片的长度可能大于基础切片的长度。假设基础切片是 baseSlice，使用操作符 [low,high]，有如下规则：0 <= low <= high <= cap(baseSlice)，只要上述满足这个关系，下标 low 和 high 都可以大于 len(baseSlice)。

引自：《Go语言101》

2.下面代码输出什么？

```

1 type N int
2
3 func (n N) test(){
4     fmt.Println(n)
5 }
6
7 func main() {
8     var n N = 10
9     p := &n
10
11     n++
12     f1 := n.test
13
14     n++
15     f2 := p.test
16
17     n++
18     fmt.Println(n)
19
20     f1()
21     f2()
22 }

```

参考答案及解析：13 11 12。知识点：方法值。当指针值赋值给变量或者作为函数参数传递时，会立即计算并复制该方法执行所需的接收者对象，与其绑定，以便在稍后执行时，能隐式传入接收者参数。

引自：《Go语言学习笔记》·方法

第 57 天

1.下面哪一行代码会 panic，请说明原因？

```
1 package main
2
3 func main() {
4     var x interface{}
5     var y interface{} = []int{3, 5}
6     _ = x == x
7     _ = x == y
8     _ = y == y
9 }
```

参考答案及解析：第 8 行。因为两个比较值的动态类型为同一个不可比较类型。

2.下面的代码输出什么？

```
1 var o = fmt.Print
2
3 func main() {
4     c := make(chan int, 1)
5     for range [3]struct{}{} {
6         select {
7             default:
8                 o(1)
9             case <-c:
10                o(2)
11                c = nil
12            case c <- 1:
13                o(3)
14        }
15    }
16 }
```

参考答案及解析：321。第一次循环，写操作已经准备好，执行 o(3)，输出 3；第二次，读操作准备好，执行 o(2)，输出 2 并将 c 赋值为 nil；第三次，由于 c 为 nil，走的是 default 分支，输出 1。

两题均引自：《Go语言101》

第 58 天

1.下面的代码输出什么？

```
1 type T struct {
2     x int
3     y *int
4 }
5
```

```

6 func main() {
7
8     i := 20
9     t := T{10,&i}
10
11    p := &t.x
12
13    *p++
14    *p--
15
16    t.y = p
17
18    fmt.Println(*t.y)
19 }

```

参考答案及解析：10。知识点：运算符优先级。如下规则：递增运算符 ++ 和递减运算符 -- 的优先级低于解引用运算符 * 和取址运算符 &，解引用运算符和取址运算符的优先级低于选择器 . 中的属性选择操作符。

2.下面哪一行代码会 panic，请说明原因？

```

1 package main
2
3 func main() {
4     x := make([]int, 2, 10)
5     _ = x[6:10]
6     _ = x[6:]
7     _ = x[2:]
8 }

```

参考答案：第 6 行，截取符号 [i:j]，如果 j 省略，默认是原切片或者数组的长度，x 的长度是 2，小于起始下标 6，所以 panic。

两题均引自：《Go语言101》

第 59 天

1.下面的代码输出什么？

```

1 type N int
2
3 func (n *N) test(){
4     fmt.Println(*n)
5 }
6
7 func main() {
8     var n N = 10
9     p := &n
10
11    n++

```

```

12     f1 := n.test
13
14     n++
15     f2 := p.test
16
17     n++
18     fmt.Println(n)
19
20     f1()
21     f2()
22 }

```

参考答案及解析：13 13 13。知识点：方法值。当目标方法的接收者是指针类型时，那么被复制的就是指针。

引自：《Go语言学习笔记》·方法

2.下面哪一行代码会 panic，请说明原因？

```

1 package main
2
3 func main() {
4     var m map[int]bool // nil
5     _ = m[123]
6     var p *[5]string // nil
7     for range p {
8         _ = len(p)
9     }
10    var s []int // nil
11    _ = s[:]
12    s, s[0] = []int{1, 2}, 9
13 }

```

参考答案及解析：第 12 行。因为左侧的 s[0] 中的 s 为 nil。

引自：《Go语言101》

第 60 天

1.下面哪一行代码会 panic，请说明原因？

```

1 package main
2
3 type T struct{}
4
5 func (*T) foo() {
6 }
7
8 func (T) bar() {
9 }

```

```

10
11 type S struct {
12     *T
13 }
14
15 func main() {
16     s := S{}
17     _ = s.foo
18     s.foo()
19     _ = s.bar
20 }

```

参考答案及解析：第 19 行，因为 s.bar 将被展开为 (*s.T).bar，而 s.T 是个空指针，解引用会 panic。

可以使用下面代码输出 s：

```

1 func main() {
2     s := S{}
3     fmt.Printf("%#v",s) // 输出: main.S{T:(*main.T)(nil)}
4 }

```

引自：《Go语言101》

2.下面的代码有什么问题？

```

1 type data struct {
2     sync.Mutex
3 }
4
5 func (d data) test(s string) {
6     d.Lock()
7     defer d.Unlock()
8
9     for i:=0;i<5 ;i++ {
10        fmt.Println(s,i)
11        time.Sleep(time.Second)
12    }
13 }
14
15
16 func main() {
17
18     var wg sync.WaitGroup
19     wg.Add(2)
20     var d data
21
22     go func() {

```

```

23     defer wg.Done()
24     d.test("read")
25 }()
26
27 go func() {
28     defer wg.Done()
29     d.test("write")
30 }()
31
32 wg.Wait()
33 }

```

参考答案及解析：锁失效。将 Mutex 作为匿名字段时，相关的方法必须使用指针接收者，否则会导致锁机制失效。

修复代码：

```

1 func (d *data) test(s string) { // 指针接收者
2     d.Lock()
3     defer d.Unlock()
4
5     for i:=0;i<5 ;i++ {
6         fmt.Println(s,i)
7         time.Sleep(time.Second)
8     }
9 }

```

或者可以通过嵌入 *Mutex 来避免复制的问题，但需要初始化。

```

1 type data struct {
2     *sync.Mutex // *Mutex
3 }
4
5 func (d data) test(s string) { // 值方法
6     d.Lock()
7     defer d.Unlock()
8
9     for i := 0; i < 5; i++ {
10        fmt.Println(s, i)
11        time.Sleep(time.Second)
12    }
13 }
14
15 func main() {
16
17    var wg sync.WaitGroup
18    wg.Add(2)
19

```

```

20  d := data{new(sync.Mutex)} // 初始化
21
22  go func() {
23      defer wg.Done()
24      d.test("read")
25  }()
26
27  go func() {
28      defer wg.Done()
29      d.test("write")
30  }()
31
32  wg.Wait()
33 }

```

引自：《Go 语言学习笔记》·同步

第 61 天

1.下面这段代码输出什么？

```

1  func main() {
2      var k = 1
3      var s = []int{1, 2}
4      k, s[k] = 0, 3
5      fmt.Println(s[0] + s[1])
6  }

```

参考答案及解析：4。知识点：多重赋值。

多重赋值分为两个步骤，有先后顺序：

- 计算等号左边的索引表达式和取址表达式，接着计算等号右边的表达式；
- 赋值；

所以本例，会先计算 `s[k]`，等号右边是两个表达式是常量，所以赋值运算等同于 `k, s[0] = 0, 3`。

2.下面代码输出什么？

```

1  func main() {
2      var k = 9
3      for k = range []int{} {}
4      fmt.Println(k)
5
6      for k = 0; k < 3; k++ {
7          }
8      fmt.Println(k)
9
10
11     for k = range (*[3]int)(nil) {

```

```
12     }
13     fmt.Println(k)
14 }
```

参考答案及解析：932。

第 62 天

1.下面哪一行代码会 panic，请说明。

```
1 func main() {
2     nil := 123
3     fmt.Println(nil)
4     var _ map[string]int = nil
5 }
```

参考答案及解析：第 4 行，当前作用域中，预定义的 nil 被覆盖，此时 nil 是 int 类型值，不能赋值给 map 类型。

2.下面代码输出什么？

```
1 func main() {
2     var x int8 = -128
3     var y = x/-1
4     fmt.Println(y)
5 }
```

参考答案及解析：-128。因为溢出。

第 63 天

1.下面选项正确的是？

- A. 类型可以声明的函数体内；
- B. Go 语言支持 ++i 或者 --i 操作；
- C. nil 是关键字；
- D. 匿名函数可以直接赋值给一个变量或者直接执行；

参考答案及解析：AD。

2.下面的代码输出什么？

```
1 func F(n int) func() int {
2     return func() int {
3         n++
4         return n
5     }
6 }
7
```

```
8 func main() {
9     f := F(5)
10    defer func() {
11        fmt.Println(f())
12    }()
13    defer fmt.Println(f())
14    i := f()
15    fmt.Println(i)
16 }
```

参考答案及解析：768。知识点：匿名函数、defer()。defer() 后面的函数如果带参数，会优先计算参数，并将结果存储在栈中，到真正执行 defer() 的时候取出。

引自《Go语言101》

第 64 天

1. 下面列举的是 recover() 的几种调用方式，哪些是正确的？

- A.

```
1 func main() {
2     recover()
3     panic(1)
4 }
```

- B.

```
1 func main() {
2     defer recover()
3     panic(1)
4 }
```

- C.

```
1 func main() {
2     defer func() {
3         recover()
4     }()
5     panic(1)
6 }
```

- D.

```

1 func main() {
2     defer func() {
3         defer func() {
4             recover()
5         }()
6     }()
7     panic(1)
8 }

```

参考答案及解析：C。recover() 必须在 defer() 函数中直接调用才有效。上面其他几种情况调用都是无效的：直接调用 recover()、在 defer() 中直接调用 recover() 和 defer() 调用时多层嵌套。

2.下面代码输出什么，请说明？

```

1 func main() {
2     defer func() {
3         fmt.Print(recover())
4     }()
5     defer func() {
6         defer fmt.Print(recover())
7         panic(1)
8     }()
9     defer recover()
10    panic(2)
11 }

```

参考答案及解析：21。recover() 必须在 defer() 函数中调用才有效，所以第 9 行代码捕获是无效的。在调用 defer() 时，便会计算函数的参数并压入栈中，所以执行第 6 行代码时，此时便会捕获 panic(2)；此后的 panic(1)，会被上一层的 recover() 捕获。所以输出 21。

引自《Go语言101》

第 65 天

1.flag 是 bool 型变量，下面 if 表达式符合编码规范的是？

- A. if flag == 1
- B. if flag
- C. if flag == false
- D. if !flag

参考答案及解析：BCD。

2.下面的代码输出什么，请说明？

```

1 func main() {
2     defer func() {
3         fmt.Print(recover())
4     }()
5     defer func() {
6         defer func() {
7             fmt.Print(recover())
8         }()
9         panic(1)
10    }()
11    defer recover()
12    panic(2)
13 }

```

参考答案及解析：12。解析请看前一天的题目。

第 66 天

1.下面的代码输出什么？

```

1 type T struct {
2     n int
3 }
4
5 func main() {
6     ts := [2]T{}
7     for i, t := range ts {
8         switch i {
9             case 0:
10                t.n = 3
11                ts[1].n = 9
12            case 1:
13                fmt.Print(t.n, " ")
14            }
15        }
16    fmt.Print(ts)
17 }

```

参考答案及解析：0 [0] {9}。知识点：for-range 循环数组。此时使用的是数组 ts 的副本，所以 t.n = 3 的赋值操作不会影响原数组。

2.下面的代码输出什么？

```

1 type T struct {
2     n int
3 }
4
5 func main() {

```

```

6   ts := [2]T{}
7   for i, t := range &ts {
8       switch i {
9           case 0:
10          t.n = 3
11          ts[1].n = 9
12          case 1:
13              fmt.Print(t.n, " ")
14          }
15      }
16      fmt.Print(ts)
17  }

```

参考答案及解析：9 [0] {9}。知识点：for-range 数组指针。for-range 循环中的循环变量 t 是原数组元素的副本。如果数组元素是结构体值，则副本的字段和原数组字段是两个不同的值。

第 67 天

1. 下面的代码输出什么？

```

1   type T struct {
2       n int
3   }
4
5   func main() {
6       ts := [2]T{}
7       for i := range ts[:] {
8           switch i {
9               case 0:
10              ts[1].n = 9
11              case 1:
12                  fmt.Print(ts[i].n, " ")
13              }
14          }
15          fmt.Print(ts)
16      }

```

参考答案及解析：9 [0] {9}。知识点：for-range 切片。for-range 切片时使用的是切片的副本，但不会复制底层数组，换句话说，此副本切片与原数组共享底层数组。

2. 下面的代码输出什么？

```

1   type T struct {
2       n int
3   }
4
5   func main() {
6       ts := [2]T{}
7       for i := range ts[:] {

```

```

8     switch t := &ts[i]; i {
9     case 0:
10        t.n = 3;
11        ts[1].n = 9
12    case 1:
13        fmt.Print(t.n, " ")
14    }
15 }
16 fmt.Print(ts)
17 }

```

参考答案及解析：9 [3] {9}。知识点：for-range 切片。

第 68 天

1.下面代码有什么问题吗？

```

1 func main() {
2
3     for i:=0;i<10 ;i++ {
4     loop:
5         println(i)
6     }
7     goto loop
8 }

```

参考答案及解析：goto 不能跳转到其他函数或者内层代码。编译报错：

```

1 goto loop jumps into block starting at

```

2.下面代码输出什么，请说明。

```

1 func main() {
2     x := []int{0, 1, 2}
3     y := [3]*int{}
4     for i, v := range x {
5         defer func() {
6             print(v)
7         }()
8         y[i] = &v
9     }
10    print(*y[0], *y[1], *y[2])
11 }

```

参考答案及解析：22222。知识点：defer()、for-range。for-range 虽然使用的是 :=，但是 v 不会重新声明，可以打印 v 的地址验证下。

第 69 天

1.关于 slice 或 map 操作, 下面正确的是?

- A

```
1 | var s []int
2 | s = append(s,1)
```

- B

```
1 | var m map[string]int
2 | m["one"] = 1
```

- C

```
1 | var s []int
2 | s = make([]int, 0)
3 | s = append(s,1)
```

- D

```
1 | var m map[string]int
2 | m = make(map[string]int)
3 | m["one"] = 1
```

参考答案及解析: ACD。

2.下面代码输出什么?

```
1 | func test(x int) (func(), func()) {
2 |     return func() {
3 |         println(x)
4 |         x += 10
5 |     }, func() {
6 |         println(x)
7 |     }
8 | }
9 |
10 | func main() {
11 |     a, b := test(100)
12 |     a()
13 |     b()
14 | }
```

参考答案及解析: 100 110。知识点: 闭包引用相同变量。

第 70 天

1.关于字符串连接，下面语法正确的是？

- A. str := 'abc' + '123'
- B. str := "abc" + "123"
- C. str := '123' + "abc"
- D. fmt.Sprintf("abc%d", 123)

参考答案及解析：BD。知识点：单引号、双引号和字符串连接。在 Go 语言中，双引号用来表示字符串 string，其实质是一个 byte 类型的数组，单引号表示 rune 类型。

2.下面代码能编译通过吗？可以的话，输出什么？

```
1 func main() {
2
3     println(DeferTest1(1))
4     println(DeferTest2(1))
5 }
6
7 func DeferTest1(i int) (r int) {
8     r = i
9     defer func() {
10        r += 3
11    }()
12    return r
13 }
14
15 func DeferTest2(i int) (r int) {
16     defer func() {
17        r += i
18    }()
19    return 2
20 }
```

参考答案及解析：43。具体解析请看《5 年 Gopher 都不知道的 defer 细节，你别再掉进坑里！》。引自：<https://my.oschina.net/qiangmzxs/blog/1515173>

第 71 天

1.判断题：对变量x的取反操作是 ~x？

从参考答案及解析：错。Go 语言的取反操作是 ^，它返回一个每个 bit 位都取反的数。作用类似在 C、C#、Java 语言中中符号 ~，对于有符号的整数来说，是按照补码进行取反操作的（快速计算方法：对数 a 取反，结果为 -(a+1)），对于无符号整数来说就是按位取反。

2.下面代码输出什么，请说明原因。

```

1  type Slice []int
2
3  func NewSlice() Slice {
4      return make(Slice, 0)
5  }
6  func (s *Slice) Add(elem int) *Slice {
7      *s = append(*s, elem)
8      fmt.Print(elem)
9      return s
10 }
11 func main() {
12     s := NewSlice()
13     defer s.Add(1).Add(2)
14     s.Add(3)
15 }

```

参考答案及解析：132。这一题有两点需要注意：1.Add() 方法的返回值依然是指针类型 *Slice，所以可以循环调用方法 Add(); 2.defer 函数的参数（包括接受者）是在 defer 语句出现的位置做计算的，而不是在函数正在执行的时候计算的，所以 s.Add(1) 会先于 s.Add(3) 执行。

第 72 天

1.下面的代码输出什么，请说明。

```

1  type Slice []int
2
3  func NewSlice() Slice {
4      return make(Slice, 0)
5  }
6  func (s *Slice) Add(elem int) *Slice {
7      *s = append(*s, elem)
8      fmt.Print(elem)
9      return s
10 }
11 func main() {
12     s := NewSlice()
13     defer func() {
14         s.Add(1).Add(2)
15     }()
16     s.Add(3)
17 }

```

参考答案及解析：312。对比昨天的第二题，本题的 s.Add(1).Add(2) 作为一个整体包在一个匿名函数中，会延迟执行。

2.下面的代码输出什么，请说明？

```

1 type Orange struct {
2     Quantity int
3 }
4
5 func (o *Orange) Increase(n int) {
6     o.Quantity += n
7 }
8
9 func (o *Orange) Decrease(n int) {
10    o.Quantity -= n
11 }
12
13 func (o *Orange) String() string {
14    return fmt.Sprintf("%#v", o.Quantity)
15 }
16
17 func main() {
18    var orange Orange
19    orange.Increase(10)
20    orange.Decrease(5)
21    fmt.Println(orange)
22 }

```

参考答案及解析：{5}。这道题容易忽视的点是，String()是指针方法，而不是值方法，所以使用Println()输出时不会调用到String()方法。

可以这样修复：

```

1 func main() {
2     orange := &Orange{}
3     orange.Increase(10)
4     orange.Decrease(5)
5     fmt.Println(orange)
6 }

```

第 73 天

1.下面代码输出什么？

```

1 func test() []func() {
2     var funs []func()
3     for i := 0; i < 2; i++ {
4         funs = append(funs, func() {
5             println(&i, i)
6         })
7     }
8     return funs
9 }

```

```

10
11 func main() {
12     funs := test()
13     for _, f := range funs {
14         f()
15     }
16 }

```

参考答案及解析：

```

1 0xc000018058 2
2 0xc000018058 2

```

知识点：闭包延迟求值。for 循环局部变量 i，匿名函数每一次使用的都是同一个变量。

如果想要匿名函数每一次输出不同，应该怎么修改代码？

引自：<https://my.oschina.net/qiangmzxs/blog/1533839>

2.下面的代码能编译通过吗？可以的话输出什么，请说明？

```

1 var f = func(i int) {
2     print("x")
3 }
4
5 func main() {
6     f := func(i int) {
7         print(i)
8         if i > 0 {
9             f(i - 1)
10        }
11    }
12    f(10)
13 }

```

参考答案及解析：10x。这道题一眼看上去会输出 109876543210，其实这是错误的答案，这里不是递归。假设 main() 函数里为 f2(), 外面的为 f1(), 当声明 f2() 时，调用的是已经完成声明的 f1()。

看下面这段代码你应该会更容易理解一点：

```

1 var x = 23
2
3 func main() {
4     x := 2*x - 4
5     println(x)    // 输出:42
6 }

```

第 74 天

1.下面代码有什么问题，请说明？

```
1 func main() {
2     runtime.GOMAXPROCS(1)
3
4     go func() {
5         for i:=0;i<10 ;i++ {
6             fmt.Println(i)
7         }
8     }()
9
10    for {}
11 }
```

参考答案及解析：for {} 独占 CPU 资源导致其他 Goroutine 饿死。

可以通过阻塞的方式避免 CPU 占用，修复代码：

```
1 func main() {
2     runtime.GOMAXPROCS(1)
3
4     go func() {
5         for i:=0;i<10 ;i++ {
6             fmt.Println(i)
7         }
8         os.Exit(0)
9     }()
10
11    select {}
12 }
```

引自《Go语言高级编程》

2.假设 x 已声明，y 未声明，下面 4 行代码哪些是正确的。错误的请说明原因？

```
1 x, _ := f() // 1
2 x, _ = f() // 2
3 x, y := f() // 3
4 x, y = f() // 4
```

参考答案及解析：2、3正确。知识点：简短变量声明。使用简短变量声明有几个需要注意的地方：

- 只能用于函数内部；
- 短变量声明语句中至少要声明一个新的变量；

第 75 天

1.下面的代码有什么问题，请说明？

```
1 func main() {
2     f, err := os.Open("file")
3     defer f.Close()
4     if err != nil {
5         return
6     }
7
8     b, err := ioutil.ReadAll(f)
9     println(string(b))
10 }
```

参考答案及解析：defer 语句应该放在 if() 语句后面，先判断 err，再 defer 关闭文件句柄。

修复代码：

```
1 func main() {
2     f, err := os.Open("file")
3     if err != nil {
4         return
5     }
6     defer f.Close()
7
8     b, err := ioutil.ReadAll(f)
9     println(string(b))
10 }
```

2.下面代码输出什么，为什么？

```
1 func f() {
2     defer func() {
3         if r := recover(); r != nil {
4             fmt.Printf("recover:%#v", r)
5         }
6     }()
7     panic(1)
8     panic(2)
9 }
10
11 func main() {
12     f()
13 }
```

参考答案及解析：recover:1。知识点：panic、recover()。当程序 panic 时就不会往下执行，可以使用 recover() 捕获 panic 的内容。

第 76 天

1.下面这段代码输出什么?

```
1  type S1 struct{}
2
3  func (s1 S1) f() {
4      fmt.Println("S1.f()")
5  }
6  func (s1 S1) g() {
7      fmt.Println("S1.g()")
8  }
9
10 type S2 struct {
11     S1
12 }
13
14 func (s2 S2) f() {
15     fmt.Println("S2.f()")
16 }
17
18 type I interface {
19     f()
20 }
21
22 func printType(i I) {
23
24     fmt.Printf("%T\n", i)
25     if s1, ok := i.(S1); ok {
26         s1.f()
27         s1.g()
28     }
29     if s2, ok := i.(S2); ok {
30         s2.f()
31         s2.g()
32     }
33 }
34
35 func main() {
36     printType(S1{})
37     printType(S2{})
38 }
```

参考答案及解析:

```
1 main.S1
2 S1.f()
3 S1.g()
4 main.S2
5 S2.f()
6 S1.g()
```

知识点：类型断言，结构体嵌套。结构体 S2 嵌套了结构体 S1，S2 自己没有实现 g()，调用的是 S1 的 g()。

2.下面的代码有什么问题？

```
1 func main() {
2     var wg sync.WaitGroup
3     wg.Add(1)
4     go func() {
5         fmt.Println("1")
6         wg.Done()
7         wg.Add(1)
8     }()
9     wg.Wait()
10 }
```

参考答案及解析：协程里面，使用 wg.Add(1) 但是没有 wg.Done()，导致 panic()。

第 77 天

1.关于 cap 函数适用下面哪些类型？

- A. 数组；
- B. channel；
- C. map；
- D. slice；

参考答案即解析：ABD。cap() 函数的作用：

- arry 返回数组的元素个数；
- slice 返回 slice 的最大容量；
- channel 返回 channel 的容量；

2.下面代码输出什么？

```
1 func hello(num ...int) {
2     num[0] = 18
3 }
4
5 func Test13(t *testing.T) {
6     i := []int{5, 6, 7}
7     hello(i...)
```

```
8   fmt.Println(i[0])
9   }
10
11  func main() {
12      t := &testing.T{}
13      Test13(t)
14  }
```

- A. 18
- B. 5
- C. Compilation error

参考答案及解析：A。可变函数是指针传递。

第 78 天

1.关于 switch 语句，下面说法正确的是？

- A. 单个 case 中，可以出现多个结果选项；
- B. 需要使用 break 来明确退出一个 case；
- C. 只有在 case 中明确添加 fallthrough 关键字，才会继续执行紧跟的下一个 case；
- D. 条件表达式必须为常量或者整数；

参考答案及解析：AC。

2.下面代码能编译通过吗？可以的话，输出什么？

```
1  func alwaysFalse() bool {
2      return false
3  }
4
5  func main() {
6      switch alwaysFalse()
7      {
8          case true:
9              println(true)
10         case false:
11             println(false)
12         }
13     }
```

参考答案及解析：可以编译通过，输出：true。知识点：Go 代码断行规则。

详情请查看：<https://gfw.go101.org/article/line-break-rules.html>

第 79 天

1.interface{} 是可以指向任意对象的 Any 类型，是否正确？

- A. false
- B. true

参考答案及解析：B。

2.下面的代码有什么问题？

```
1  type ConfigOne struct {
2      Daemon string
3  }
4
5  func (c *ConfigOne) String() string {
6      return fmt.Sprintf("print: %v", c)
7  }
8
9  func main() {
10     c := &ConfigOne{}
11     c.String()
12 }
```

参考答案及解析：无限递归循环，栈溢出。知识点：类型的 String() 方法。如果类型定义了 String() 方法，使用 Printf()、Print()、Println()、Sprintf() 等格式化输出时会自动使用 String() 方法。

第 80 天

1.定义一个包内全局字符串变量，下面语法正确的是？

- A. var str string
- B. str := ""
- C. str = ""
- D. var str = ""

参考答案及解析：AD。全局变量要定义在函数之外，而在函数之外定义的变量只能用 var 定义。短变量声明 := 只能用于函数之内。

2.下面的代码有什么问题？

```
1  func main() {
2
3      wg := sync.WaitGroup{}
4
5      for i := 0; i < 5; i++ {
```

```

6     go func(wg sync.WaitGroup, i int) {
7         wg.Add(1)
8         fmt.Printf("i:%d\n", i)
9         wg.Done()
10        }(wg, i)
11    }
12
13    wg.Wait()
14
15    fmt.Println("exit")
16 }

```

参考答案及解析：知识点：WaitGroup 的使用。存在两个问题：

- 在协程中使用 wg.Add();
- 使用了 sync.WaitGroup 副本；

修复代码：

```

1  func main() {
2
3      wg := sync.WaitGroup{}
4
5      for i := 0; i < 5; i++ {
6          wg.Add(1)
7          go func(i int) {
8              fmt.Printf("i:%d\n", i)
9              wg.Done()
10             }(i)
11        }
12
13        wg.Wait()
14
15        fmt.Println("exit")
16    }

```

或者：

```

1  func main() {
2
3      wg := &sync.WaitGroup{}
4
5      for i := 0; i < 5; i++ {
6          wg.Add(1)
7          go func(wg *sync.WaitGroup, i int) {
8              fmt.Printf("i:%d\n", i)
9              wg.Done()
10             }(wg, i)
11        }

```

```
12
13     wg.Wait()
14
15     fmt.Println("exit")
16 }
```

第 81 天

1. 下面的代码输出什么？

```
1 func main() {
2     var a []int = nil
3     a, a[0] = []int{1, 2}, 9
4     fmt.Println(a)
5 }
```

参考答案即解析：运行时错误。知识点：多重赋值。

多重赋值分为两个步骤，有先后顺序：

- 计算等号左边的索引表达式和取址表达式，接着计算等号右边的表达式；
- 赋值；

2. 下面代码中的指针 p 为野指针，因为返回的栈内存在函数结束时会被释放？

```
1 type TimesMatcher struct {
2     base int
3 }
4
5 func NewTimesMatcher(base int) *TimesMatcher {
6     return &TimesMatcher{base:base}
7 }
8
9 func main() {
10    p := NewTimesMatcher(3)
11    fmt.Println(p)
12 }
```

- A. false
- B. true

参考答案及解析：A。Go 语言的内存回收机制规定，只要有一个指针指向引用一个变量，那么这个变量就不会被释放（内存逃逸），因此在 Go 语言中返回函数参数或临时变量是安全的。

第 82 天

1.下面这段代码输出什么？

```
1 func main() {
2     count := 0
3     for i := range [256]struct{}{} {
4         m, n := byte(i), int8(i)
5         if n == -n {
6             count++
7         }
8         if m == -m {
9             count++
10        }
11    }
12    fmt.Println(count)
13 }
```

参考答案及解析：4。知识点：数值溢出。当 i 的值为 0、128 是会发生相等情况，注意 byte 是 uint8 的别名。

引自《Go语言101》

2.下面代码输出什么？

```
1 const (
2     azero = iota
3     aone  = iota
4 )
5
6 const (
7     info = "msg"
8     bzero = iota
9     bone  = iota
10 )
11
12 func main() {
13     fmt.Println(azero, aone)
14     fmt.Println(bzero, bone)
15 }
```

参考答案及解析：0 1 1 2。知识点：iota 的使用。这道题易错点在 bzero、bone 的值，在一个常量声明代码块中，如果 iota 没出现在第一行，则常量的初始值就是非 0 值。

第 83 天

1. 同级文件的包名不允许有多个，是否正确？

- A. true
- B. false

参考答案及解析：A。一个文件夹下只能有一个包，可以多个.go文件，但这些文件必须属于同一个包。

2. 下面的代码有什么问题，请说明。

```
1  type data struct {
2      name string
3  }
4
5  func (p *data) print() {
6      fmt.Println("name:", p.name)
7  }
8
9  type printer interface {
10     print()
11 }
12
13 func main() {
14     d1 := data{"one"}
15     d1.print()
16
17     var in printer = data{"two"}
18     in.print()
19 }
```

参考答案及解析：编译报错。

```
1  cannot use data literal (type data) as type printer in assignment:
2  data does not implement printer (print method has pointer receiver)
```

结构体类型 data 没有实现接口 printer。知识点：接口。

第 84 天

1. 函数执行时，如果由于 panic 导致了异常，则延迟函数不会执行。这一说法是否正确？

- A. true
- B. false

参考答案及解析：B。由 panic 引发异常以后，程序停止执行，然后调用延迟函数（defer），就像程序正常退出一样。

2.下面代码输出什么？

```
1 func main() {
2     a := [3]int{0, 1, 2}
3     s := a[1:2]
4
5     s[0] = 11
6     s = append(s, 12)
7     s = append(s, 13)
8     s[0] = 21
9
10    fmt.Println(a)
11    fmt.Println(s)
12 }
```

参考答案及解析：

输出：

```
1 [0 11 12]
2 [21 12 13]
```

第 85 天

1.下面这段代码输出什么？请简要说明。

```
1 func main() {
2     fmt.Println(strings.TrimRight("ABBA", "BA"))
3 }
```

参考答案及解析：输出空字符。这是一个大多数人遇到的坑，TrimRight() 会将第二个参数字符串里面所有的字符拿出来处理，只要与其中任何一个字符相等，便会将其删除。想正确地截取字符串，可以参考 TrimSuffix() 函数。

2.下面代码输出什么？

```
1 func main() {
2     var src, dst []int
3     src = []int{1, 2, 3}
4     copy(dst, src)
5     fmt.Println(dst)
6 }
```

参考答案及解析：输出 []。知识点：拷贝切片。copy(dst, src) 函数返回 len(dst)、len(src) 之间的最小值。如果要将 src 完全拷贝至 dst，必须给 dst 分配足够的内存空间。

修复代码：

```

1 func main() {
2     var src, dst []int
3     src = []int{1, 2, 3}
4     dst = make([]int, len(src))
5     n := copy(dst, src)
6     fmt.Println(n,dst)
7 }

```

或者直接使用 append()

```

1 func main() {
2     var src, dst []int
3     src = []int{1, 2, 3}
4     dst = append(dst, src...)
5     fmt.Println("dst:", dst)
6 }

```

第 86 天

1.n 是秒数，下面代码输出什么？

```

1 func main() {
2     n := 43210
3     fmt.Println(n/60*60, " hours and ", n%60*60, " seconds")
4 }

```

参考答案及解析：43200 hours and 600 seconds。知识点：运算符优先级。算术运算符 *、/ 和 % 的优先级相同，从左向右结合。

修复代码如下：

```

1 func main() {
2     n := 43210
3     fmt.Println(n/(60*60), "hours and", n%(60*60), "seconds")
4 }

```

2.下面代码输出什么，为什么？

```

1 const (
2     Century = 100
3     Decade  = 010
4     Year    = 001
5 )
6
7 func main() {
8     fmt.Println(Century + 2*Decade + 2*Year)
9 }

```

参考答案及解析：118。知识点：进制数。Go 语言里面，八进制数以 0 开头，十六进制数以 0x 开头，所以 Decade 表示十进制的 8。

第 87 天

1.关于协程，下面说法正确是()

- A.协程和线程都可以实现程序的并发执行；
- B.线程比协程更轻量级；
- C.协程不存在死锁问题；
- D.通过 channel 来进行协程间的通信；

参考答案及解析：AD。

2.在数学里面，有著名的勾股定理：

```
1 | a^2+b^2=c^2
```

例如，有我们熟悉的组合 (3, 4, 5)、(6、8、10) 等。在 Go 语言中，下面代码输出 true：

```
1 | fmt.Println(3^2+4^2 == 5^2) // true
```

问题来了，下面代码输出什么，请简要说明。

```
1 | func main() {  
2 |     fmt.Println(6^2+8^2 == 10^2)  
3 | }
```

参考答案及解析：false。在 Go 语言里面，^ 作为二元运算符时表示按位异或：对应位，相同为 0，相异为 1。所以第一段代码输出 true 是因为：

```
1 | 0011 ^ 0010 == 0001    (3^2 == 1)  
2 | 0100 ^ 0010 == 0110    (4^2 == 6)  
3 | 0101 ^ 0010 == 0111    (5^2 == 7)
```

1+6=7，这当然是相等的。你来试试分解下第二段代码的数学表达式！

参考：<https://yourbasic.org/golang/gotcha-bitwise-operators/>

第 88 天

1.下面这段代码能通过编译吗？请简要说明。

```
1 func main() {
2     m := make(map[string]int)
3     m["foo"]++
4     fmt.Println(m["foo"])
5 }
```

参考答案及解析：能通过编译。

上面的代码可以理解成：

```
1 func main() {
2     m := make(map[string]int)
3     v := m["foo"]
4     v++
5     m["foo"] = v
6     fmt.Println(m["foo"])
7 }
```

2.下面的代码输出什么，请简要说明？

```
1 func Foo() error {
2     var err *os.PathError = nil
3     // ...
4     return err
5 }
6
7 func main() {
8     err := Foo()
9     fmt.Println(err)
10    fmt.Println(err == nil)
11 }
```

参考答案及解析：nil false。知识点：接口值与 nil 值。只有在值和动态类型都为 nil 的情况下，接口值才为 nil。Foo() 函数返回的 err 变量，值为 nil、动态类型为 *os.PathError，与 nil（值为 nil，动态类型为 nil）显然是不相等。我们可以打印下变量 err 的详情：

```
1 fmt.Printf("%#v\n",err) // (*os.PathError)(nil)
```

一个更合适的解决办法：

```

1 func Foo() (err error) {
2     // ...
3     return
4 }
5
6 func main() {
7     err := Foo()
8     fmt.Println(err)
9     fmt.Println(err == nil)
10 }

```

第 89 天

1.下面代码能编译通过吗？请简要说明。

```

1 func main() {
2     v := []int{1, 2, 3}
3     for i, n := 0, len(v); i < n; i++ {
4         v = append(v, i)
5     }
6     fmt.Println(v)
7 }

```

参考答案及解析：能编译通过，输出 [1 2 3 0 1 2]。for range 循环开始的时候，终止条件只会计算一次。

2.下面代码输出什么？

```

1 type P *int
2 type Q *int
3
4 func main() {
5     var p P = new(int)
6     *p += 8
7     var x *int = p
8     var q Q = x
9     *q++
10    fmt.Println(*p, *q)
11 }

```

- A.8 8
- B.8 9
- C.9 9

参考答案及解析：C。指针变量指向相同的地址。

第 90 天

1. 下面代码能通过编译吗？

```
1 type T int
2
3 func F(t T) {}
4
5 func main() {
6     var q int
7     F(q)
8 }
```

2. 下面代码能通过编译吗？请简要说明。

```
1 type T []int
2
3 func F(t T) {}
4
5 func main() {
6     var q []int
7     F(q)
8 }
```

我们将这两道题目放到一块做一个解析，第一题不能通过编译，第二题可以通过编译。我们知道不同类型的值是不能相互赋值的，即使底层类型一样，所以第一题编译不通过；对于底层类型相同的变量可以相互赋值还有一个重要的条件，即至少有一个不是有名类型（named type）。

这是 Go 语言规范手册的原文：

```
"x's type V and T have identical underlying types and at least one of V or T is not a named type."
```

Named Type 有两类：

- 内置类型，比如 int, int64, float, string, bool 等；
- 使用关键字 type 声明的类型；

Unnamed Type 是基于已有的 Named Type 组合一起的类型，例如：struct{}、[]string、interface{}、map[string]bool 等。

引自：https://mp.weixin.qq.com/s?_biz=MzA3MjIwNzYyNA==&mid=2650918436&idx=1&sn=094e9c52fe922415a64297a18fcc1bb4&chksm=84d4b243b3a33b55aa6bb99c495fe4156659c1ac1e0d8312b232054b51a26f041a32b4929f4a#rd

第 91 天

1.下面两段代码能否编译通过？请简要说明。

第一段：

```
1 func f() {}
2 func f() {}
3
4 func main() {}
```

第二段：

```
1 func init(){}
2 func init(){}
3
4 func main() {}
```

参考答案及解析：第二段代码能通过编译。除 `init()` 函数之外，一个包内不允许有其他同名函数。

2.下面代码有什么问题？请指出。

```
1 func (m map[string]string) Set(key string, value string) {
2     m[key] = value
3 }
4
5 func main() {
6     m := make(map[string]string)
7     m.Set("A", "One")
8 }
```

参考答案及解析：Unnamed Type 不能作为方法的接收者。昨天我们讲过 Named Type 与 Unamed Type 的区别，就用 Named Type 来修复下代码：

```
1 type User map[string]string
2
3 func (m User) Set(key string, value string) {
4     m[key] = value
5 }
6
7 func main() {
8     m := make(User)
9     m.Set("A", "One")
10 }
```

第 92 天

1.下面代码输出什么?

```
1  var x int
2
3  func init() {
4      x++
5  }
6
7  func main() {
8      init()
9      fmt.Println(x)
10 }
```

参考答案及解析：编译失败。init() 函数不能被其他函数调用，包括 main() 函数。

2.min() 函数是求两个数之间的较小值，能否在 该函数中添加一行代码将其功能补全。

```
1  func min(a int, b uint) {
2      var min = 0
3      fmt.Printf("The min of %d and %d is %d\n", a, b, min)
4  }
5
6  func main() {
7      min(1225, 256)
8  }
```

参考答案即解析：利用 copy() 函数的功能：切片复制，并且返回两者长度的较小值。

```
1  func min(a int, b uint) {
2      var min = 0
3      min = copy(make([]struct{}),a),make([]struct{}),b))
4      fmt.Printf("The min of %d and %d is %d\n", a, b, min)
5  }
6
7  func main() {
8      min(1225, 256)
9  }
```

第 93 天

1.关于 main() 函数，下面说法正确的是?

- 不能带参数;
- 不能定义返回值;
- 所在的包必须为 main 包;
- 可以使用 flag 包来获取和解析命令行参数;

参考答案及解析：ABCD。

2.下面代码能编译通过吗？请简要说明。

```
1 type User struct {
2     Name string
3 }
4
5 func (u *User) SetName(name string) {
6     u.Name = name
7     fmt.Println(u.Name)
8 }
9
10 type Employee User
11
12 func main() {
13     employee := new(Employee)
14     employee.SetName("Jack")
15 }
```

参考答案及解析：编译不通过。当使用 type 声明一个新类型，它不会继承原有类型的方法集。

第 94 天

1.下面这段代码输出什么？请简要说明。

```
1 func main() {
2     a := 2 ^ 15
3     b := 4 ^ 15
4     if a > b {
5         println("a")
6     } else {
7         println("b")
8     }
9 }
```

参考答案及解析：a。Go 语言里面 ^ 表示按位异或，而不是求幂。

```
1 0010 ^ 1111 == 1101 (2^15 == 13)
2 0100 ^ 1111 == 1011 (4^15 == 11)
```

2.下面哪些函数不能通过编译？

```
1 func A(string string) string {
2     return string + string
3 }
```

```

4
5 func B(len int) int {
6     return len + len
7 }
8
9 func C(val, default string) string {
10    if val == "" {
11        return default
12    }
13    return val
14 }

```

参考答案及解析：C() 函数不能通过编译。C() 函数的 default 属于关键字。string 和 len 是预定义标识符，可以在局部使用。nil 也可以当做变量使用，不过不建议写这样的代码，可读性不好，小心被接手你代码的人胖揍。

```

1 var nil = new(int)
2
3 func main() {
4     var p *int
5     if p == nil {
6         fmt.Println("p is nil")
7     } else {
8         fmt.Println("p is not nil")
9     }
10 }

```

第 95 天

1.下面代码输出什么？请简要说明。

```

1 type foo struct{ Val int }
2
3 type bar struct{ Val int }
4
5 func main() {
6     a := &foo{Val: 5}
7     b := &foo{Val: 5}
8     c := foo{Val: 5}
9     d := bar{Val: 5}
10    e := bar{Val: 5}
11    f := bar{Val: 5}
12    fmt.Print(a == b, c == foo(d), e == f)
13 }

```

参考答案及解析：false true true。这道题唯一有疑问的地方就在第一个比较，Go 语言里没有引用变量，每个变量都占用一个惟一的内存位置，所以第一个比较输出 false。

2.下面代码输出什么？

```

1 func A() int {
2     time.Sleep(100 * time.Millisecond)
3     return 1
4 }
5
6 func B() int {
7     time.Sleep(1000 * time.Millisecond)
8     return 2
9 }
10
11 func main() {
12     ch := make(chan int, 1)
13     go func() {
14         select {
15             case ch <- A():
16             case ch <- B():
17             default:
18                 ch <- 3
19         }
20     }()
21     fmt.Println(<-ch)
22 }

```

参考答案及解析：1、2随机输出。

第 96 天

1. 下面的代码输出什么？

```

1 type Point struct{ x, y int }
2
3 func main() {
4     s := []Point{
5         {1, 2},
6         {3, 4},
7     }
8     for _, p := range s {
9         p.x, p.y = p.y, p.x
10    }
11    fmt.Println(s)
12 }

```

参考答案及解析：输出 [{1 2} {3 4}]。知识点：for range 循环。range 循环的时候，获取到的元素值是一个副本，就比如这里的 p。修复代码示例：

```

1 type Point struct{ x, y int }
2
3 func main() {
4     s := []*Point{
5         &Point{1, 2},
6         &Point{3, 4},
7     }
8     for _, p := range s {
9         p.x, p.y = p.y, p.x
10    }
11    fmt.Println(*s[0])
12    fmt.Println(*s[1])
13 }

```

2.下面的代码有什么隐患?

```

1 func get() []byte {
2     raw := make([]byte, 10000)
3     fmt.Println(len(raw), cap(raw), &raw[0])
4     return raw[:3]
5 }
6
7 func main() {
8     data := get()
9     fmt.Println(len(data), cap(data), &data[0])
10 }

```

参考答案及解析: get() 函数返回的切片与原切片公用底层数组, 如果在调用函数里面 (这里是 main() 函数) 修改返回的切片, 将会影响到原切片。为了避免掉入陷阱, 可以如下修改:

```

1 func get() []byte {
2     raw := make([]byte, 10000)
3     fmt.Println(len(raw), cap(raw), &raw[0])
4     res := make([]byte, 3)
5     copy(res, raw[:3])
6     return res
7 }
8
9 func main() {
10    data := get()
11    fmt.Println(len(data), cap(data), &data[0])
12 }

```

第 97 天

1.关于map，下面说法正确的是？

- A. map 反序列化时 json.Unmarshal() 的入参必须为map的地址；
- B. 在函数调用中传递 map，则子函数中对 map 元素的增加不会导致父函数中 map 的修改；
- C. 在函数调用中传递 map，则子函数中对 map 元素的修改不会导致父函数中 map 的修改；
- D. 不能使用内置函数 delete() 删除 map 的元素；

参考答案及解析：A。知识点：map 的使用。

2.下面代码输出什么？请简要说明。

```
1 type Foo struct {
2     val int
3 }
4
5 func (f Foo) Inc(inc int) {
6     f.val += inc
7 }
8
9 func main() {
10    var f Foo
11    f.Inc(100)
12    fmt.Println(f.val)
13 }
```

参考答案及解析：输出 0。使用值类型接收者定义的方法，调用的时候，使用的是值的副本，对副本操作不会影响的原来的值。如果想要在调用函数中修改原值，可以使用指针接收者定义的方法。

```
1 type Foo struct {
2     val int
3 }
4
5 func (f *Foo) Inc(inc int) {
6     f.val += inc
7 }
8
9 func main() {
10    f := &Foo{}
11    f.Inc(100)
12    fmt.Println(f.val) // 100
13 }
```

第 98 天

1.下面代码输出什么?

```
1 func main() {
2     a := 1
3     for i := 0;i<5;i++ {
4         a := a + 1
5         a = a * 2
6     }
7     fmt.Println(a)
8 }
```

参考答案及解析: 1。知识点: 变量的作用域。注意 for 语句的变量 a 是重新声明, 它的作用范围只在 for 语句范围内。

2.下面的代码输出什么?

```
1 func test(i int) (ret int) {
2     ret = i * 2
3     if ret > 10 {
4         ret := 10
5         return
6     }
7     return
8 }
9
10 func main() {
11     result := test(10)
12     fmt.Println(result)
13 }
```

参考答案即解析: 编译错误。知识点: 变量的作用域。编译错误信息: ret is shadowed during return。

第 99 天

1.下面代码能编译通过吗?

```
1 func main() {
2     true := false
3     fmt.Println(true)
4 }
```

参考答案即解析: 编译通过。true 是预定义标识符可以用作变量名, 但是不建议这么做。

2.下面的代码输出什么?

```
1 func watShadowDefer(i int) (ret int) {
```

```

2   ret = i * 2
3   if ret > 10 {
4       ret := 10
5       defer func() {
6           ret = ret + 1
7       }()
8   }
9   return
10  }
11
12  func main() {
13      result := watShadowDefer(50)
14      fmt.Println(result)
15  }

```

参考答案即解析：100。知识点：变量作用域和defer 返回值。

第 100 天

1.下面代码输出什么？

```

1   func main() {
2       m := map[string]int{
3           "G": 7, "A": 1,
4           "C": 3, "E": 5,
5           "D": 4, "B": 2,
6           "F": 6, "I": 9,
7           "H": 8,
8       }
9       var order []string
10      for k, _ := range m {
11          order = append(order, k)
12      }
13      fmt.Println(order)
14  }

```

参考答案即解析：按字母无序输出。知识点：遍历 map 是无序的。

2.下面的代码有什么问题？

```

1   type UserAges struct {
2       ages map[string]int
3       sync.Mutex
4   }
5
6   func (ua *UserAges) Add(name string, age int) {
7       ua.Lock()
8       defer ua.Unlock()
9       ua.ages[name] = age

```

```

10 }
11
12 func (ua *UserAges) Get(name string) int {
13     if age, ok := ua.ages[name]; ok {
14         return age
15     }
16     return -1
17 }
18
19 func main() {
20     count := 1000
21     gw := sync.WaitGroup{}
22     gw.Add(count * 3)
23     u := UserAges{ages: map[string]int{}}
24     add := func(i int) {
25         u.Add(fmt.Sprintf("user_%d", i), i)
26         gw.Done()
27     }
28     for i := 0; i < count; i++ {
29         go add(i)
30         go add(i)
31     }
32     for i := 0; i < count; i++ {
33         go func(i int) {
34             defer gw.Done()
35             u.Get(fmt.Sprintf("user_%d", i))
36         }(i)
37     }
38     gw.Wait()
39     fmt.Println("Done")
40 }

```

参考答案即解析：在执行 Get() 方法时可能报错。知识点：读写锁。虽然可以使用 sync.Mutex 做写锁，但是 map 是并发读写不安全的。map 属于引用类型，并发读写时多个协程是通过指针访问同一个地址，即访问共享变量，此时同时读写资源存在竞争关系，会报错“fatal error: concurrent map read and map write”。

有兴趣的同学可以尝试用 sync.RWMutex（读写锁）改进下程序。

引自：<https://juejin.im/entry/5971bed66fb9a06bb21adf15>

第 101 天

1.关于循环语句，下面说法正确的有？

- A. 循环语句既支持 for 关键字，也支持 while 和 do-while；
- B. 关键字for的基本使用方法与C/C++中没有任何差异；
- C. for 循环支持 continue 和 break 来控制循环，但是它提供了一个更高级的 break，可以选择中断哪一个循环；
- D. for 循环不支持以逗号为间隔的多个赋值语句，必须使用平行赋值的方式来初始化多个变量；

参考答案及解析：CD。

引自：<https://blog.csdn.net/fhd994603831/article/details/90648525>

2.下面代码的功能是从小到大找出 17 和 38 的 3 个公倍数，请问下面的代码有什么问题？

```
1  var ch chan int = make(chan int)
2
3  func generate() {
4      for i := 17; i < 5000; i += 17 {
5          ch <- i
6          time.Sleep(1 * time.Millisecond)
7      }
8      close(ch)
9  }
10
11 func main() {
12     timeout := time.After(800 * time.Millisecond)
13     go generate()
14     found := 0
15     for {
16         select {
17             case i, ok := <-ch:
18                 if ok {
19                     if i%38 == 0 {
20                         fmt.Println(i, "is a multiple of 17 and 38")
21                         found++
22                         if found == 3 {
23                             break
24                         }
25                     }
26                 } else {
27                     break
28                 }
29             case <-timeout:
30                 fmt.Println("timed out")
31                 break
32         }
33     }
34     fmt.Println("The end")
35 }
```

参考答案即解析：break 会跳出 select 块，但不会跳出 for 循环。这算是一个比较容易掉的坑。可以使用 break label 特性或者 goto 功能解决这个问题，这里使用 break label 作个示例。

```
1  var ch chan int = make(chan int)
2
```

```

3 func generate() {
4     for i := 17; i < 5000; i += 17 {
5         ch <- i
6         time.Sleep(1 * time.Millisecond)
7     }
8     close(ch)
9 }
10
11 func main() {
12     timeout := time.After(800 * time.Millisecond)
13     go generate()
14     found := 0
15     MAIN_LOOP:
16     for {
17         select {
18         case i, ok := <-ch:
19             if ok {
20                 if i%38 == 0 {
21                     fmt.Println(i, "is a multiple of 17 and 38")
22                     found++
23                     if found == 3 {
24                         break MAIN_LOOP
25                     }
26                 }
27             } else {
28                 break MAIN_LOOP
29             }
30         case <-timeout:
31             fmt.Println("timed out")
32             break MAIN_LOOP
33         }
34     }
35     fmt.Println("The end")
36 }

```

第 102 天

1.ch := make(chan interface{}) 和 ch := make(chan interface{},1) 有什么区别?

参考答案及解析：第一个是声明无缓存通道，第二个是声明缓存为 1 的通道。无缓存通道需要一直有接收者接收数据，写操作才会继续，不然会一直阻塞；而缓冲为 1 则即使没有接收者也不会阻塞，因为缓冲大小是 1，只有当放第二个值的时候，第一个还没被人拿走，这时候才会阻塞。注意这两者还是有区别的。

2.下面的代码输出什么？请简要说明。

```

1 var mu sync.Mutex
2 var chain string
3
4 func main() {

```

```

5   chain = "main"
6   A()
7   fmt.Println(chain)
8   }
9   func A() {
10  mu.Lock()
11  defer mu.Unlock()
12  chain = chain + " --> A"
13  B()
14  }
15
16  func B() {
17  chain = chain + " --> B"
18  C()
19  }
20
21  func C() {
22  mu.Lock()
23  defer mu.Unlock()
24  chain = chain + " --> C"
25  }

```

- A. 不能编译;
- B. 输出 main --> A --> B --> C;
- C. 输出 main;
- D. fatal error;

参考答案即解析：D。使用 Lock() 加锁后，不能再继续对其加锁，直到利用 Unlock() 解锁后才能再加锁。

引自博客《鸟窝》 <https://colobu.com/>

第 103 天

1.下面代码输出什么?

```

1   func main() {
2   fmt.Println(doubleScore(0))
3   fmt.Println(doubleScore(20.0))
4   fmt.Println(doubleScore(50.0))
5   }
6   func doubleScore(source float32) (score float32) {
7   defer func() {
8       if score < 1 || score >= 100 {
9           score = source
10          }
11      }()
12      return source * 2
13  }

```

参考答案及解析：输出 0 40 50。知识点：defer 语句与返回值。函数的 return value 不是原子操作，而是在编译器中分解为两部分：返回值赋值 和 return。

2.下面代码输出什么？请简要说明。

```
1 var mu sync.RWMutex
2 var count int
3
4 func main() {
5     go A()
6     time.Sleep(2 * time.Second)
7     mu.Lock()
8     defer mu.Unlock()
9     count++
10    fmt.Println(count)
11 }
12 func A() {
13     mu.RLock()
14     defer mu.RUnlock()
15     B()
16 }
17 func B() {
18     time.Sleep(5 * time.Second)
19     C()
20 }
21 func C() {
22     mu.RLock()
23     defer mu.RUnlock()
24 }
```

- A. 不能编译；
- B. 输出 1；
- C. 程序 hang 住；
- D. fatal error；

参考答案及解析：D。当写锁阻塞时，新的读锁是无法申请的（有效防止写锁饥饿），导致死锁。

第 104 天

1.关于同步锁，下面说法正确的是？

- A. 当一个 goroutine 获得了 Mutex 后，其他 goroutine 就只能乖乖的等待，除非该 goroutine 释放这个 Mutex；
- B. RWMutex 在读锁占用的情况下，会阻止写，但不阻止读；
- C. RWMutex 在写锁占用情况下，会阻止任何其他 goroutine（无论读和写）进来，整个锁相当于由该 goroutine 独占；
- D. Lock() 操作需要保证有 Unlock() 或 RUnlock() 调用与之对应；

参考答案及解析：ABC。

2.下面代码输出什么？请简要说明。

```
1 func main() {
2     var wg sync.WaitGroup
3     wg.Add(1)
4     go func() {
5         time.Sleep(time.Millisecond)
6         wg.Done()
7         wg.Add(1)
8     }()
9     wg.Wait()
10 }
```

- A. 不能编译；
- B. 无输出，正常退出；
- C. 程序 hang 住；
- D. panic；

参考答案及解析：D。WaitGroup 在调用 Wait() 之后不能再调用 Add() 方法的。

第 105 天

1.下面代码输出什么？请简要说明。

```
1 var c = make(chan int)
2 var a int
3
4 func f() {
5     a = 1
6     <-c
7 }
8 func main() {
9     go f()
10    c <- 0
11    print(a)
12 }
```

- A. 不能编译；
- B. 输出 1；
- C. 输出 0；
- D. panic；

参考答案及解析：B。能正确输出，不过主协程会阻塞 f() 函数的执行。

2.下面代码输出什么？请简要说明。

```
1 type MyMutex struct {
```

```

2   count int
3   sync.Mutex
4 }
5
6 func main() {
7     var mu MyMutex
8     mu.Lock()
9     var mu1 = mu
10    mu.count++
11    mu.Unlock()
12    mu1.Lock()
13    mu1.count++
14    mu1.Unlock()
15    fmt.Println(mu.count, mu1.count)
16 }

```

- A. 不能编译;
- B. 输出 1, 1;
- C. 输出 1, 2;
- D. fatal error;

参考答案及解析：D。加锁后复制变量，会将锁的状态也复制，所以 mu1 其实是已经加锁状态，再加锁会死锁。

第 106 天

1.下面代码输出什么？请简要说明。

```

1 func main() {
2     var ch chan int
3     var count int
4     go func() {
5         ch <- 1
6     }()
7     go func() {
8         count++
9         close(ch)
10    }()
11    <-ch
12    fmt.Println(count)
13 }

```

- A. 不能编译;
- B. 输出 1;
- C. 输出 0;
- D. panic;

参考答案及解析：D。ch 未有被初始化，关闭时会报错。

2.下面代码输出什么？请简要说明。

```

1 func main() {
2     var ch chan int
3     go func() {
4         ch = make(chan int, 1)
5         ch <- 1
6     }()
7     go func(ch chan int) {
8         time.Sleep(time.Second)
9         <-ch
10    }(ch)
11
12    c := time.Tick(1 * time.Second)
13    for range c {
14        fmt.Printf("#goroutines: %d\n", runtime.NumGoroutine())
15    }
16 }

```

- A. 不能编译;
- B. 一段时间后总是输出 #goroutines: 1;
- C. 一段时间后总是输出 #goroutines: 2;
- D. panic;

参考答案即解析：C。程序执行到第二个 goroutine 时，ch 还未初始化，导致第二个 goroutine 阻塞。需要注意的是第一个 goroutine 不会阻塞。

引自博客《鸟窝》 <https://colobu.com/>

第 107 天

1. 下面代码输出什么？请简要说明。

```

1 func main() {
2     var m sync.Map
3     m.LoadOrStore("a", 1)
4     m.Delete("a")
5     fmt.Println(m.Len())
6 }

```

- A. 不能编译;
- B. 输出 1;
- C. 输出 0;
- D. panic;

参考答案及解析：D。sync.Map 没有 Len() 方法。

2. 下面代码输出什么？请简要说明。

```

1 func main() {
2     var wg sync.WaitGroup
3     wg.Add(2)
4     var ints = make([]int, 0, 1000)
5     go func() {
6         for i := 0; i < 1000; i++ {
7             ints = append(ints, i)
8         }
9         wg.Done()
10    }()
11    go func() {
12        for i := 0; i < 1000; i++ {
13            ints = append(ints, i)
14        }
15        wg.Done()
16    }()
17    wg.Wait()
18    fmt.Println(len(ints))
19 }

```

- A. 不能编译;
- B. 输出 2000;
- C. 输出可能不是 2000;
- D. panic;

参考答案及解析：C。append() 并不是并发安全的，有兴趣的同学可以尝试用锁去解决这个问题。

第 108 天

1. 下面的代码输出什么？

```

1 type People struct {
2     name string `json:"name"`
3 }
4
5 func main() {
6     js := `{
7         "name": "11"
8     }`
9     var p People
10    err := json.Unmarshal([]byte(js), &p)
11    if err != nil {
12        fmt.Println("err: ", err)
13        return
14    }
15    fmt.Println("people: ", p)
16 }

```

参考答案及解析：people:{}. 按照 go 的语法，小写开头的方法、属性或 struct 是私有的，同样，在 json 解码或转码的时候也无法实现私有属性的转换。

这段代码是无法正常得到 People 的 name 值的。而且，私有属性 name 也不应该加 json 的标签。

2.补充 A、B 两处代码，实现程序能解析 ip 和 port 参数，默认值是 0.0.0.0 和 8000。

```
1  var ip string
2  var port int
3
4  func init() {
5      // A
6      // B
7  }
8
9  func main() {
10     flag.Parse()
11     fmt.Printf("%s:%d", ip, port)
12 }
```

参考答案及解析：flag 包的使用。

```
1  var ip string
2  var port int
3
4  func init() {
5     flag.StringVar(&ip, "ip", "0.0.0.0", "ip address")
6     flag.IntVar(&port, "port", 8000, "port number")
7  }
8
9  func main() {
10     flag.Parse()
11     fmt.Printf("%s:%d", ip, port)
12 }
```

第 109 天

1.下面代码有什么问题？

```
1  func main() {
2     ch := make(chan int, 1000)
3     go func() {
4         for i := 0; i < 10; i++ {
5             ch <- i
6         }
7     }()
8     go func() {
9         for {
```

```

10     a, ok := <-ch
11     if !ok {
12         fmt.Println("close")
13         return
14     }
15     fmt.Println("a: ", a)
16 }
17 }()
18 close(ch)
19 fmt.Println("ok")
20 time.Sleep(time.Second * 20)
21 }

```

参考答案及解析：panic。协程开启还未来得及执行，chan 就已经 close()，往已经关闭的 chan 写数据会 panic。

2.在 A 处添加一行代码实现 S 按升序排列。

```

1 type S struct {
2     v int
3 }
4
5 func main() {
6     s := []S{{1}, {3}, {5}, {2}}
7     // A
8     fmt.Printf("%#v", s)
9 }

```

参考答案及解析：可以考虑使用 sort.Slice()。

```

1 type S struct {
2     v int
3 }
4
5 func main() {
6     s := []S{{1}, {3}, {5}, {2}}
7     sort.Slice(s, func(i, j int) bool { return s[i].v < s[j].v })
8     fmt.Printf("%#v", s)
9 }

```

第 110 天

1.下面代码输出什么？请简要说明。

```

1 type T struct {
2     v int
3 }

```

```

4
5 func (t *T) Incr(wg *sync.WaitGroup) {
6     t.V++
7     wg.Done()
8 }
9 func (t *T) Print() {
10    time.Sleep(1)
11    fmt.Print(t.V)
12 }
13 func main() {
14    var wg sync.WaitGroup
15    wg.Add(10)
16    var ts = make([]T, 10)
17    for i := 0; i < 10; i++ {
18        ts[i] = T{i}
19    }
20    for _, t := range ts {
21        go t.Incr(&wg)
22    }
23    wg.Wait()
24    for _, t := range ts {
25        go t.Print()
26    }
27    time.Sleep(5 * time.Second)
28 }

```

- A. 输出 12345678910;
- B. 输出 0123456789;
- C. 输出 9999999999;
- D. panic;

参考答案及解析：C。这道题需要注意的一点是 for range 循环里的变量 t 是临时变量。

2.下面的代码可以随机输出大小写字母，尝试在 A 处添加一行代码使得字母先按大写再按小写的顺序输出。

```

1  const N = 26
2
3  func main() {
4      const GOMAXPROCS = 1
5      runtime.GOMAXPROCS(GOMAXPROCS)
6
7      var wg sync.WaitGroup
8      wg.Add(2 * N)
9      for i := 0; i < N; i++ {
10         go func(i int) {
11             defer wg.Done()

```

```

12     // A
13     runtime.Gosched()
14     fmt.Printf("%c", 'a'+i)
15 } (i)
16 go func(i int) {
17     defer wg.Done()
18     fmt.Printf("%c", 'A'+i)
19 } (i)
20 }
21 wg.Wait()
22 }

```

参考答案及解析:

```

1  const N = 26
2
3  func main() {
4      const GOMAXPROCS = 1
5      runtime.GOMAXPROCS(GOMAXPROCS)
6
7      var wg sync.WaitGroup
8      wg.Add(2 * N)
9      for i := 0; i < N; i++ {
10         go func(i int) {
11             defer wg.Done()
12             runtime.Gosched()
13             fmt.Printf("%c", 'a'+i)
14         } (i)
15         go func(i int) {
16             defer wg.Done()
17             fmt.Printf("%c", 'A'+i)
18         } (i)
19     }
20     wg.Wait()
21 }

```

第 111 天

1. 下面两处打印的值是否相同? 请简要说明。

```

1 func main() {
2     var val int
3     println(&val)
4     f(10000)
5     println(&val)
6 }
7
8 func f(i int) {
9     if i--; i == 0 {
10        return
11    }
12    f(i)
13 }

```

参考答案及解析：不同。知识点：栈增长、逃逸分析。每个 goroutine 都会分配相应的栈内存，比如 Go 1.11 版本是 2Kb，随着程序运行，栈内存会发生增长或缩小，协程会重新申请栈内存块。就像这个题目，循环调用 f()，发生深度递归，栈内存不断增大，当超过范围时，会重新申请栈内存，所以 val 的地址会变化。

这道题还有个特别注意的地方，如果将 println() 函数换成 fmt.Println() 会发现，打印结果相同。为什么？因为函数 fmt.Println() 使变量 val 发生了逃逸，逃逸到堆内存，即使协程栈内存重新申请，val 变量在堆内存的地址也不会改变。

2.下面代码 A 处输出什么？请简要说明。

```

1 func main() {
2     var val int
3
4     a := &val
5     println(a)
6
7     f(10000)
8
9     b := &val
10    println(b)
11
12    println(a == b) // A
13 }
14
15 func f(i int) {
16     if i--; i == 0 {
17        return
18    }
19    f(i)
20 }

```

- A. true
- B. false

参考答案及解析：A。这道题和上一道有一定联系，a 是指向变量 val 的指针，我们知道 val 变量的地址发生了改变，a 指向 val 新的地址是由内存管理自动实现的。

```
1 func main() {
2     var val int
3
4     a := &val
5     println(a)
6
7     f(10000)
8
9     b := &val
10    println(a) // a b 的值相同
11    println(b)
12
13    println(a == b) // A
14 }
15
16 func f(i int) {
17     if i--; i == 0 {
18         return
19     }
20     f(i)
21 }
```

来源：<https://twitter.com/empijei/status/1206718810025267200>

相关阅读:

<https://utcc.utoronto.ca/~cks/space/blog/programming/GoPointerToInteger>

<https://blog.cloudflare.com/how-stacks-are-handled-in-go/amp/>

第 112 天

1.下面代码输出什么？请简要说明。

```
1 func main() {
2     x := []int{100, 200, 300, 400, 500, 600, 700}
3     twohundred := &x[1]
4     x = append(x, 800)
5     for i := range x {
6         x[i]++
7     }
8     fmt.Println(*twohundred)
9 }
```

参考答案及解析：200。因为原切片的容量已经满了，执行 append 操作之后会创建一个新的底层数组，并将原切片底层数组的值拷贝到新的数组，原数组保持不变。

```
1 func main() {
2
3     x := make([]int, 0, 7)
4     x = append(x, 100, 200, 300, 400, 500, 600, 700)
5     twohundred := &x[1]
6     x = append(x, 800)
7     for i := range x {
8         x[i]++
9     }
10    fmt.Println(*twohundred)    // 输出 200
11
12    x = make([]int, 0, 8)    // 指向另一个切片
13    x = append(x, 100, 200, 300, 400, 500, 600, 700)
14    twohundred = &x[1]
15    x = append(x, 800)    // 执行 append 操作，容量足够，不会重新申请内存
16    for i := range x {
17        x[i]++
18    }
19    fmt.Println(*twohundred)    // 输出 201
20 }
21
```

2.下面的代码输出什么？请简要说明。

```
1 func main() {
2     a := []int{0, 1}
3     fmt.Printf("%v", a[len(a):])
4 }
```

参考答案及解析：输出 []。对一个切片执行 [i:j] 的时候，i 和 j 都不能超过切片的长度值。

第 113 天

1.关于 const 常量定义，下面正确的使用方式是？

A.

```
1 const Pi float64 = 3.14159265358979323846
2 const zero= 0.0
```

B.

```
1 const (  
2     size int64= 1024  
3     eof = -1  
4 )
```

C.

```
1 const (  
2     ERR_ELEM_EXISTError = errors.New("element already exists")  
3     ERR_ELEM_NT_EXISTError = errors.New("element not exists")  
4 )
```

D.

```
1 const u, vfloat32 = 0, 3  
2 const a,b, c = 3, 4, "foo"
```

参考答案及解析：ABD。

2.修改下面的代码，使得第二个输出 [seek 1 2 3 4]。

```
1 func link(p ...interface{}) {  
2     fmt.Println(p)  
3 }  
4  
5 func main() {  
6     link("seek", 1, 2, 3, 4) // 输出 [seek 1 2 3 4]  
7     a := []int{1, 2, 3, 4}  
8     link("seek", a) // 输出 [seek [1 2 3 4]]  
9 }
```

参考答案及解析：

```
1 func link(p ...interface{}) {  
2     fmt.Println(p)  
3 }  
4  
5 func main() {  
6     link("seek", 1, 2, 3, 4) // 输出 [seek 1 2 3 4]  
7     a := []int{1, 2, 3, 4}  
8     link("seek", a) // 输出 [seek [1 2 3 4]]  
9  
10    tmplink := make([]interface{}, 0, len(a)+1)  
11    tmplink = append(tmplink, "seek")  
12    for _, ii := range a {  
13        tmplink = append(tmplink, ii)  
14    }
```

```
15 link(tmplink...) // 输出 [seek 1 2 3 4]
16 }
```

第 114 天

1.下面代码输出什么？

```
1 func main() {
2     ns := []int{010: 200, 005: 100}
3     print(len(ns))
4 }
```

参考答案及解析：9。Go 语言中，0x 开头表示 十六进制；0 开头表示八进制。

2.下面的代码输出什么？请简要说明。

```
1 func main() {
2     i := 0
3     f := func() int {
4         i++
5         return i
6     }
7     c := make(chan int, 1)
8     c <- f()
9     select {
10    case c <- f():
11    default:
12        fmt.Println(i)
13    }
14 }
```

参考答案即解析：2。知识点：select 的使用。

下面这段代码会更有助于大家理解：

```
1 func main() {
2     i := 0
3     f := func() int {
4         fmt.Println("incr")
5         i++
6         return i
7     }
8     c := make(chan int)
9     for j := 0; j < 2; j++ {
10        select {
11        case c <- f():
12            // noop
```

```
13     default:
14         // noop
15     }
16 }
17 fmt.Println(i)
18 }
```

第 115 天

1.下面正确的是?

```
1  var y int
2
3  func f(x int) int {
4      return 7
5  }
6
7  A.
8  switch y = f(2) {
9      case y == 7:
10         return
11     }
12
13  B.
14  switch y = f(2); {
15      case y == 7:
16         return
17     }
18
19  C.
20  switch y = f(2) {
21      case 7:
22         return
23     }
24
25  D.
26  switch y = f(2); {
27      case 7:
28         return
29     }
```

参考答案及解析：B。知识点：switch case 的使用。

2.下面的代码输出什么？

```

1 func main() {
2     a := []int{1, 2, 3, 4}
3     b := variadic(a...)
4     b[0], b[1] = b[1], b[0]
5     fmt.Println(a)
6 }
7
8 func variadic(ints ...int) []int {
9     return ints
10 }

```

参考答案及解析：2 1 3 4。知识点：可变函数。切片作为参数传入可变函数时不会创建新的切片。

第 116 天

1.下面的代码输出什么？

```

1 const (
2     one = 1 << iota
3     two
4 )
5
6 func main() {
7     fmt.Println(one, two)
8 }

```

2.下面的代码输出什么？

```

1 const (
2     greeting = "Hello, Go"
3     one = 1 << iota
4     two
5 )
6
7 func main() {
8     fmt.Println(one, two)
9 }

```

参考答案及解析：这两道题考的是同一个知识点：iota 的使用。第一题：1 2；第二题：2, 4。

第 117 天

1.Go 语言中大多数数据类型都可以转化为有效的JSON文本，下面几种类型除外。

- A. 指针
- B. channel
- C. complex
- D. 函数

参考答案及解析：BCD。

2.下面代码输出什么？如果想要代码输出 10，应该如何修改？

```
1  const N = 10
2
3  func main() {
4      m := make(map[int]int)
5
6      wg := &sync.WaitGroup{}
7      mu := &sync.Mutex{}
8      wg.Add(N)
9      for i := 0; i < N; i++ {
10         go func() {
11             defer wg.Done()
12             mu.Lock()
13             m[i] = i
14             mu.Unlock()
15         }()
16     }
17     wg.Wait()
18     println(len(m))
19 }
```

参考答案及解析：输出 1。知识点：并发、引用。修复代码如下：

```
1  const N = 10
2
3  func main() {
4      m := make(map[int]int)
5
6      wg := &sync.WaitGroup{}
7      mu := &sync.Mutex{}
8      wg.Add(N)
9      for i := 0; i < N; i++ {
10         go func(i int) {
11             defer wg.Done()
12             mu.Lock()
13             m[i] = i
14             mu.Unlock()
15         }(i)
16     }
17     wg.Wait()
18     println(len(m))
19 }
```

第 118 天

1、下面说法正确的是。

- A. Go 语言中，声明的常量未使用会报错；
- B. cap() 函数适用于 array、slice、map 和 channel；
- C. 空指针解析会触发异常；
- D. 从一个已经关闭的 channel 接收数据，如果缓冲区中为空，则返回一个零值；

参考答案及解析：CD。A.不会报错；B.cap() 函数不适用 map。

2.下面的代码输出什么？

```
1  const (
2      _ = iota
3      c1 int = (10*iota)
4      c2
5      d = iota
6  )
7  func main() {
8      fmt.Printf("%d - %d - %d",c1,c2, d)
9  }
```

- A. compile error
- B. 1 - 2 - 3
- C. 10 - 20 - 30
- D. 10 - 20 - 3

参考答案及解析：D。iota 的使用。

相关阅读：

<https://twitter.com/gopherconf/status/1205451736678371328>

<https://github.com/golang/go/wiki/lota>

第 119 天

1.关于slice或map操作，下面正确的是。

A.

```
1  var s []int
2  s = append(s,1)
```

B.

```
1  var m map[string]int
2  m["one"] = 1
```

C.

```
1 var s []int
2 s = make([]int, 0)
3 s = append(s,1)
```

D.

```
1 var m map[string]int
2 m = make(map[string]int)
3 m["one"] = 1
```

参考答案及解析：ACD。

2.下面代码输出什么？请简要说明。

```
1 var ErrDidNotWork = errors.New("did not work")
2
3 func DoTheThing(reallyDoIt bool) (err error) {
4     if reallyDoIt {
5         result, err := tryTheThing()
6         if err != nil || result != "it worked" {
7             err = ErrDidNotWork
8         }
9     }
10    return err
11 }
12
13 func tryTheThing() (string, error) {
14    return "", ErrDidNotWork
15 }
16
17 func main() {
18    fmt.Println(DoTheThing(true))
19    fmt.Println(DoTheThing(false))
20 }
```

参考答案即解析：都输出 nil。知识点：变量的作用域。因为 if 语句块内的 err 变量会遮罩函数作用域内的 err 变量。

修复代码：

```

1 func DoTheThing(reallyDoIt bool) (err error) {
2     var result string
3     if reallyDoIt {
4         result, err = tryTheThing()
5         if err != nil || result != "it worked" {
6             err = ErrDidNotWork
7         }
8     }
9     return err
10 }

```

第 120 天

1.下面代码输出什么？

```

1 func main() {
2     fmt.Println(len("你好bj!"))
3 }

```

参考答案及解析：9。知识点：编码长度。

2.是否可以编译通过？如果通过，输出什么？

```

1 func GetValue(m map[int]string, id int) (string, bool) {
2     if _, exist := m[id]; exist {
3         return "存在数据", true
4     }
5     return nil, false
6 }
7
8 func main() {
9
10    intmap := map[int]string{
11        1: "a",
12        2: "bb",
13        3: "ccc",
14    }
15
16    v, err := GetValue(intmap, 3)
17    fmt.Println(v, err)
18 }

```

参考答案及解析：函数返回值类型 nil 可以用作 interface、function、pointer、map、slice 和 channel 的“空值”。但是如果不特别指定的话，Go 语言不能识别类型，所以会报错。通常编译的时候不会报错，但是运行是时会报:cannot use nil as type string in return argument.

引自《Go夜读》

由 SeekLoad 整理，公众号：Golang来啦。



由「Go语言中文网」同步发布：

