第 1 天

1.下面这段代码输出的内容

```go
package main

import (
    "fmt"
)

func main() {
    defer_call()
}

func defer_call() {
    defer func() { fmt.Println("打印前") }()
    defer func() { fmt.Println("打印中") }()
    defer func() { fmt.Println("打印后") }()

    panic("触发异常")
}
```

答案:

```
打印后
打印中
打印前
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。

```go
func main() {

  slice := []int{0,1,2,3}
  m := make(map[int]*int)

  for key,val := range slice {
    m[key] = &val
  }

```

```
  for k,v := range m {
    fmt.Println(k,"->",*v)
  }
}
```

参考答案：

```
0 -> 3
1 -> 3
2 -> 3
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：

```
func main() {

  slice := []int{0,1,2,3}
  m := make(map[int]*int)

  for key,val := range slice {
    value := val
    m[key] = &value
  }

  for k,v := range m {
    fmt.Println(k,"===>",*v)
  }
}
```

扩展题目

```
type Test struct {
  name string
}

func (this *Test) Point(){
  fmt.Println(this.name)
}


func main() {

  ts := []Test{
```

```
    {"a"},
    {"b"},
    {"c"},
  }

  for _,t := range ts {
    //fmt.Println(reflect.TypeOf(t))
    defer t.Point()
  }

}
```

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.
func main() {
    s := make([]int, 5)
    s = append(s, 1, 2, 3)
    fmt.Println(s)
}

// 2.
func main() {
  s := make([]int,0)
  s = append(s,1,2,3,4)
  fmt.Println(s)
}
```

2.下面这段代码有什么缺陷

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

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。

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

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?

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

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?

```
var(
  size := 1024
  max_size = size*2
)

func main() {
  fmt.Println(size,max_size)
}
```

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？

```
func main() {
  sn1 := struct {
    age  int
    name string
  }{age: 11, name: "qq"}
  sn2 := struct {
    age  int
    name string
  }{age: 11, name: "qq"}

  if sn1 == sn2 {
    fmt.Println("sn1 == sn2")
  }

  sm1 := struct {
    age int
    m   map[string]string
  }{age: 11, m: map[string]string{"a": "1"}}
  sm2 := struct {
    age int
    m   map[string]string
  }{age: 11, m: map[string]string{"a": "1"}}

  if sm1 == sm2 {
    fmt.Println("sm1 == sm2")
  }
}
```

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；

```
    sn3:= struct {
    name string
    age  int
  }{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？

```
const (
  x = iota
  _
  y
  z = "zz"
  k
  p = iota
)

func main()  {
  fmt.Println(x,y,z,k,p)
}
```

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。

```
type error interface {
  Error() string
}
```

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，不可出现死循坏。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?

```
func hello() []string {
    return nil
}

func main() {
    h := hello
    if h == nil {
        fmt.Println("nil")
    } else {
        fmt.Println("not nil")
    }
}
```

- 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?

```
func GetValue() int {
  return 1
}

func main() {
  i := GetValue()
  switch i.(type) {
  case int:
    println("int")
  case string:
    println("string")
  case interface{}:
    println("interface")
  default:
    println("unknown")
  }
}
```

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

```
type person struct {
    name string
}

func main() {
    var m map[person]int
    p := person{"mike"}
    fmt.Println(m[p])
}
```

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

```go
func hello(num ...int) {
    num[0] = 18
}

func main() {
    i := []int{5, 6, 7}
    hello(i...)
    fmt.Println(i[0])
}
```

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?

```go
func main() {
    a := 5
    b := 8.1
    fmt.Println(a + b)
}
```

- 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?

```go
package main

import (
    "fmt"
)

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

- 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？

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

- 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？

```go
func main() {
    var i interface{}
    if i == nil {
        fmt.Println("nil")
        return
    }
    fmt.Println("not nil")
}
```

- 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?

```go
func main() {
    s := make(map[string]int)
    delete(s, "h")
    fmt.Println(s["h"])
}
```

- 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；获取不存在的减值对时，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?

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

- 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?

```
type People struct{}

func (p *People) ShowA() {
  fmt.Println("showA")
  p.ShowB()
}
func (p *People) ShowB() {
  fmt.Println("showB")
}

type Teacher struct {
  People
}

func (t *Teacher) ShowB() {
  fmt.Println("teacher showB")
}

func main() {
  t := Teacher{}
  t.ShowB()
}
```

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?

```
func hello(i int) {
    fmt.Println(i)
}
func main() {
    i := 5
    defer hello(i)
    i = i + 10
}
```

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?

```
type People struct{}

func (p *People) ShowA() {
  fmt.Println("showA")
  p.ShowB()
}
func (p *People) ShowB() {
  fmt.Println("showB")
}

type Teacher struct {
  People
}

func (t *Teacher) ShowB() {
  fmt.Println("teacher showB")
}

func main() {
  t := Teacher{}
  t.ShowA()
}
```

参考答案及解析：

```
showA
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?

```
func main() {
  str := "hello"
  str[0] = 'x'
  fmt.Println(str)
}
```

- 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?

```
func incr(p *int) int {
  *p++
  return *p
}

func main() {
  p :=1
  incr(&p)
  fmt.Println(p)
}
```

- 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?

```
func add(args ...int) int {

  sum := 0
  for _, arg := range args {
    sum += arg
  }
  return sum
}
```

- 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?

```
func main() {
  var s1 []int
  var s2 = []int{}
  if __ == nil {
    fmt.Println("yes nil")
  }else{
    fmt.Println("no nil")
  }
}
```

- 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?

```
func main() {
    i := 65
    fmt.Println(string(i))
}
```

- 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?

```
type A interface {
  ShowA() int
}

type B interface {
  ShowB() int
}

```

```go
type Work struct {
  i int
}

func (w Work) ShowA() int {
  return w.i + 10
}

func (w Work) ShowB() int {
  return w.i + 20
}

func main() {
  c := Work{3}
  var a A = c
  var b B = c
  fmt.Println(a.ShowA())
  fmt.Println(b.ShowB())
}
```

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?

```go
func main() {

  s := [3]int{1, 2, 3}
  a := s[:0]
  b := s[:2]
  c := s[1:2:cap(s)]
}
```

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?

```go
func main() {
  var m map[string]int        //A
  m["a"] = 1
  if v := m["b"]; v != nil {  //B
    fmt.Println(v)
  }
}
```

参考答案及解析：

```go
func main() {
  m := make(map[string]int)
  m["a"] = 1
  if v,ok := m["b"]; ok {
    fmt.Println(v)
  }
}
```

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?

```go
type A interface {
  ShowA() int
}

type B interface {
  ShowB() int
}

type Work struct {
  i int
}

func (w Work) ShowA() int {
  return w.i + 10
}

func (w Work) ShowB() int {
  return w.i + 20
}

```

```
func main() {
  c := Work{3}
  var a A = c
  var b B = c
  fmt.Println(a.ShowB())
  fmt.Println(b.ShowA())
}
```

- 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：

```
a.ShowB undefined (type A has no field or method ShowB)
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. x, _ := f()
2. x, _ = f()
3. x, y := f()
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?

```
func increaseA() int {
  var i int
  defer func() {
    i++
  }()
  return i
}

func increaseB() (r int) {
  defer func() {
    r++
  }()
  return r
}

func main() {
```

```
  fmt.Println(increaseA())
  fmt.Println(increaseB())
}
```

- 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?

```
type A interface {
  ShowA() int
}

type B interface {
  ShowB() int
}

type Work struct {
  i int
}

func (w Work) ShowA() int {
  return w.i + 10
}

func (w Work) ShowB() int {
  return w.i + 20
}

func main() {
  var a A = Work{3}
  s := a.(Work)
  fmt.Println(s.ShowA())
  fmt.Println(s.ShowB())
}
```

- 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?

```go
func f1() (r int) {
  defer func() {
    r++
  }()
  return 0
}


func f2() (r int) {
  t := 5
  defer func() {
    t = t + 5
  }()
  return t
}


func f3() (r int) {
  defer func(r int) {
    r = r + 5
  }(r)
  return 1
}
```

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?

```go
type Person struct {
  age int
}

func main() {
  person := &Person{28}

  // 1.
  defer fmt.Println(person.age)

  // 2.
  defer func(p *Person) {
    fmt.Println(p.age)
  }(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}
```

```
  // 1.
  defer fmt.Println(person.age)

  // 2.
  defer func(p *Person) {
    fmt.Println(p.age)
  }(person)

  // 3.
  defer func() {
    fmt.Println(person.age)
  }()

  person = &Person{29}
}
```

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？

```
A. var a []int
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?

```
type S struct {
}

func f(x interface{}) {
}

```

```
func g(x *interface{}) {
}

func main() {
  s := S{}
  p := &s
  f(s) //A
  g(s) //B
  f(p) //C
  g(p) //D
}
```

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?

```
type S struct {
  m string
}

func f() *S {
  return __  //A
}

func main() {
  p := __    //B
  fmt.Println(p.m) //print "foo"
}
```

参考答案及解析：

```
A. &S{"foo"}
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?

```go
package main
import (
    "fmt"
)
func main() {
    var x string = nil
    if x == nil {
        x = "default"
    }
    fmt.Println(x)
}
```

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?

```go
var a bool = true
func main() {
  defer func(){
    fmt.Println("1")
  }()
  if a == true {
    fmt.Println("2")
    return
  }
  defer func(){
    fmt.Println("3")
  }()
}
```

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?

```go
func main() {

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

参考答案及解析：

```
[1 2 4]
[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?

```go
func main() {
  if a := 1; false {
  } else if b := 2; false {
  } else {
    println(a, b)
  }
}
```

- 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?

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

参考答案及解析：

```
0 zero
1 one
// 或者
1 one
0 zero
```

map 的输出是无序的。

2.下面这段代码输出什么?

```go
func main() {
  a := 1
  b := 2
  defer calc("1", a, calc("10", a, b))
  a = 0
  defer calc("2", a, calc("20", a, b))
  b = 1
}

func calc(index string, a, b int) int {
  ret := a + b
  fmt.Println(index, a, b, ret)
  return ret
}
```

参考答案及解析：

```
10 1 2 3
20 0 2 2
2 0 2 2
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：2022，1134。

第 25 天

1.下面这段代码输出什么？为什么？

```
func (i int) PrintInt ()  {
  fmt.Println(i)
}

func main() {
  var i int = 1
  i.PrintInt()
}
```

- 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：

```
type Myint int

func (i Myint) PrintInt ()  {
  fmt.Println(i)
}

func main() {
  var i Myint = 1
  i.PrintInt()
}
```

2.下面这段代码输出什么？为什么？

```
type People interface {
  Speak(string) string
}

type Student struct{}

func (stu *Student) Speak(think string) (talk string) {
  if think == "speak" {
    talk = "speak"
  } else {
    talk = "hi"
  }
```

```
  return
}

func main() {
  var peo People = Student{}
  think := "speak"
  fmt.Println(peo.Speak(think))
}
```

- 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` 实现改方法。

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?

```
const (
  a = iota
  b = iota
)
const (
  name = "name"
  c    = iota
  d    = iota
)
func main() {
  fmt.Println(a)
  fmt.Println(b)
  fmt.Println(c)
  fmt.Println(d)
}
```

参考答案及解析：0 1 1 2。知识点：iota 的用法。

iota 是 golang 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？

```go
type People interface {
  Show()
}

type Student struct{}

func (stu *Student) Show() {

}

func main() {

  var s *Student
  if s == nil {
    fmt.Println("s is nil")
  } else {
    fmt.Println("s is not nil")
  }
  var p People = s
  if p == nil {
    fmt.Println("p is nil")
  } else {
    fmt.Println("p is not nil")
  }
}
```

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？

```go
type Direction int

const (
  North Direction = iota
  East
  South
  West
)

func (d Direction) String() string {
  return [...]string{"North", "East", "South", "West"}[d]
```

```
}

func main() {
  fmt.Println(South)
}
```

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?

```
type Math struct {
  x, y int
}

var m = map[string]Math{
  "foo": Math{2, 3},
}

func main() {
  m["foo"].x = 4
  fmt.Println(m["foo"].x)
}
```

- 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

```
type Math struct {
  x, y int
}

var m = map[string]Math{
  "foo": Math{2, 3},
```

```
}

func main() {
  tmp := m["foo"]
  tmp.x = 4
  m["foo"] = tmp
  fmt.Println(m["foo"].x)
}
```

b.修改数据结构

```
type Math struct {
  x, y int
}

var m = map[string]*Math{
  "foo": &Math{2, 3},
}

func main() {
  m["foo"].x = 4
  fmt.Println(m["foo"].x)
  fmt.Printf("%#v", m["foo"])    // %#v 格式化输出详细信息
}
```

references:

0.https://blog.csdn.net/qq_36431213/article/details/82805043
1.https://www.cnblogs.com/DillGao/p/7930674.html
2.https://haobook.readthedocs.io/zh_CN/latest/periodical/201611/zhangan.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?

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

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?

```go
var p *int

func foo() (*int, error) {
  var i int = 5
  return &i, nil
}

func bar() {
  //use p
  fmt.Println(*p)
}

func main() {
  p, err := foo()
  if err != nil {
    fmt.Println(err)
    return
  }
  bar()
  fmt.Println(*p)
}
```

- 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：

```go
func main() {
  var err error
  p, err = foo()
  if err != nil {
    fmt.Println(err)
    return
  }
  bar()
  fmt.Println(*p)
}
```

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?

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

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?

```
func main() {

  var m = [...]int{1, 2, 3}

  for i, v := range m {
    go func() {
      fmt.Println(i, v)
    }()
  }

  time.Sleep(time.Second * 3)
}
```

参考答案及解析：

```
2 3
2 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

```
for i, v := range m {
  go func(i,v int) {
    fmt.Println(i, v)
  }(i,v)
}
```

b.使用临时变量保留当前值

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

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

第 30 天

1.下面这段代码输出什么?

```
func f(n int) (r int) {
  defer func() {
    r += n
    recover()
  }()

  var f func()

  defer f()
  f = func() {
    r += 2
  }
  return n + 1
}

func main() {
  fmt.Println(f(3))
}
```

参考答案及解析：7。

2.下面这段代码输出什么?

```
func main() {
```

```
  var a = [5]int{1, 2, 3, 4, 5}
  var r [5]int

  for i, v := range a {
    if i == 0 {
      a[1] = 12
      a[2] = 13
    }
    r[i] = v
  }
  fmt.Println("r = ", r)
  fmt.Println("a = ", a)
}
```

参考答案及解析：

```
r =  [1 2 3 4 5]
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：

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

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：

```
func main() {
  var a = [5]int{1, 2, 3, 4, 5}
  var r [5]int

  for i, v := range &a {
    if i == 0 {
      a[1] = 12
      a[2] = 13
    }
    r[i] = v
  }
  fmt.Println("r = ", r)
  fmt.Println("a = ", a)
```

```
}
```

输出：

```
r =  [1 12 13 4 5]
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-attation-to-in-go/

第 31 天

1.下面这段代码输出什么?

```
func change(s ...int) {
  s = append(s,3)
}

func main() {
  slice := make([]int,5,5)
  slice[0] = 1
  slice[1] = 2
  change(slice...)
  fmt.Println(slice)
  change(slice[0:2]...)
  fmt.Println(slice)
}
```

参考答案及解析：

```
[1 2 0 0 0]
[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?

```
func main() {
  var a = []int{1, 2, 3, 4, 5}
  var r [5]int

  for i, v := range a {
```

```
    if i == 0 {
      a[1] = 12
      a[2] = 13
    }
    r[i] = v
  }
  fmt.Println("r = ", r)
  fmt.Println("a = ", a)
}
```

参考答案及解析：

```
r =  [1 12 13 4 5]
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-attation-to-in-go/

第 32 天

1.下面这段代码输出结果正确吗?

```
type Foo struct {
  bar string
}
func main() {
  s1 := []Foo{
    {"A"},
    {"B"},
    {"C"},
  }
  s2 := make([]*Foo, len(s1))
  for i, value := range s1 {
    s2[i] = &value
  }
  fmt.Println(s1[0], s1[1], s1[2])
  fmt.Println(s2[0], s2[1], s2[2])
}
```

```
输出:
{A} {B} {C}
&{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：

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

2.下面代码里的 counter 的输出值?

```
func main() {

  var m = map[string]int{
    "A": 21,
    "B": 22,
    "C": 23,
  }
  counter := 0
  for k, v := range m {
    if counter == 0 {
      delete(m, "A")
    }
    counter++
    fmt.Println(k, v)
  }
  fmt.Println("counter is ", counter)
}
```

- 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?

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

- 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?

```
A.
type MyInt int
var i int = 1
var j MyInt = i

B.
type MyInt int
var i int = 1
var j MyInt = (MyInt)i

C.
type MyInt int
var i int = 1
var j MyInt = MyInt(i)

```

```
D.
type MyInt int
var i int = 1
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：

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

则Add函数定义正确的是()

```
A.
type Integer int
func (a Integer) Add(b Integer) Integer {
        return a + b
}

B.
type Integer int
func (a Integer) Add(b *Integer) Integer {
        return a + *b
}

C.
type Integer int
func (a *Integer) Add(b Integer) Integer {
        return *a + b
}

D.
type Integer int
func (a *Integer) Add(b *Integer) Integer {
```

```
        return *a + *b
}
```

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?

```
A.
i := 1
i++

B.
i := 1
j = i++

C.
i := 1
++i

D.
i := 1
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

```
type Fragment interface {
        Exec(transInfo *TransInfo) error
}
type GetPodAction struct {
}
func (g GetPodAction) Exec(transInfo *TransInfo) error {
        ...
        return nil
}
```

- 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。

```
func main() {
  runtime.GOMAXPROCS(1)
  int_chan := make(chan int, 1)
  string_chan := make(chan string, 1)
  int_chan <- 1
  string_chan <- "hello"
  select {
  case value := <-int_chan:
    fmt.Println(value)
  case value := <-string_chan:
    panic(value)
  }
}
```

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?

```
const i = 100
var j = 123

func main() {
  fmt.Println(&j, j)
  fmt.Println(&i, i)
}
```

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?

```
func GetValue(m map[int]string, id int) (string, bool) {

  if _, exist := m[id]; exist {
    return "exist", true
  }
  return nil, false
}
func main() {
  intmap := map[int]string{
    1: "a",
    2: "b",
    3: "c",
  }

  v, err := GetValue(intmap, 3)
  fmt.Println(v, err)
}
```

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?

```
func main() {
  x := []string{"a", "b", "c"}
  for v := range x {
    fmt.Print(v)
  }
}
```

参考答案及解析：012。注意区别下面代码段：

```
func main() {
  x := []string{"a", "b", "c"}
  for _, v := range x {
    fmt.Print(v)     //输出 abc
  }
}
```

3.下面这段代码能否编译通过？如果通过，输出什么?

```
type User struct{}
type User1 User
type User2 = User

func (i User1) m1() {
  fmt.Println("m1")
}
func (i User) m2() {
  fmt.Println("m2")
}

func main() {
  var i1 User1
  var i2 User2
  i1.m1()
  i2.m2()
}
```

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?

```
func Foo(x interface{}) {
  if x == nil {
    fmt.Println("empty interface")
    return
  }
  fmt.Println("non-empty interface")
}
func main() {
  var x *int = nil
  Foo(x)
}
```

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?

```
func main() {
  ch := make(chan int, 100)
  // A
  go func() {
    for i := 0; i < 10; i++ {
      ch <- i
    }
  }()
  // B
  go func() {
    for {
      a, ok := <-ch
      if !ok {
        fmt.Println("close")
```

```
        return
      }
      fmt.Println("a: ", a)
    }
  }()
  close(ch)
  fmt.Println("ok")
  time.Sleep(time.Second * 10)
}
```

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. golang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?

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

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?

```
type Param map[string]interface{}

type Show struct {
  *Param
}

func main() {
  s := new(Show)
  s.Param["day"] = 2
}
```

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：

```go
func main() {
  s := new(Show)
  // 修复代码
  p := make(Param)
  p["day"] = 2
  s.Param = &p
  tmp := *s.Param
  fmt.Println(tmp["day"])
}
```

第 41 天

1.下面代码编译能通过吗?

```go
func main()
{
    fmt.Println("hello world")
}
```

参考答案及解析：编译错误。

```
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：

```go
func main() {
  fmt.Println("works")
}
```

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

2.下面这段代码输出什么?

```go
var x = []int{2: 2, 3, 0: 1}

func main() {
  fmt.Println(x)
}
```

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?

```go
func incr(p *int) int {
  *p++
  return *p
}
func main() {
  v := 1
  incr(&v)
  fmt.Println(v)
}
```

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?

```go
package main

var gvar int

func main() {
    var one int
    two := 2
    var three int
    three = 3

    func(unused string) {
        fmt.Println("Unused arg. No compile error")
    }("what?")
}
```

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：

```go
func main() {
  var one int
  _ = one

  two := 2
  fmt.Println(two)
```

```
  var three int
  three = 3
  one = three

  var four int
  four = four
}
```

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?

```
type ConfigOne struct {
  Daemon string
}

func (c *ConfigOne) String() string {
  return fmt.Sprintf("print: %v", c)
}

func main() {
  c := &ConfigOne{}
  c.String()
}
```

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。

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

3.下面代码输出什么?

```
func main() {
  var a = []int{1, 2, 3, 4, 5}
  var r = make([]int, 0)

  for i, v := range a {
    if i == 0 {
      a = append(a, 6, 7)
    }

    r = append(r, v)
  }

```

```
    fmt.Println(r)
}
```

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?

```
import (
    "fmt"
    "log"
    "time"
)
func main() {
}
```

参考答案及解析：导入的包没有被使用。如果引入一个包，但是未使用其中如何函数、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：

```
import (
    _ "fmt"
    "log"
    "time"
)
var _ = log.Println
func main() {
    _ = time.Now
}
```

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

2.下面代码输出什么?

```go
func main() {
    x := interface{}(nil)
    y := (*int)(nil)
    a := y == x
    b := y == nil
    _, c := x.(interface{})
    println(a, b, c)
}
```

- 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。

```go
func main() {

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

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

参考答案及解析：有 1 出错误，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：

```go
func main() {
  var m map[string]int
  m = make(map[string]int)
  m["one"] = 1
}
```

第 44 天

1.下面代码有什么问题?

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

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?

```
func main() {
    var x = nil
    _ = x
}
```

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：

```
func main() {
  var x interface{} = nil
  _ = x
}
```

3.下面代码能编译通过吗?

```
type info struct {
  result int
}

func work() (int,error) {
  return 13,nil
}

func main() {
  var data info

  data.result, err := work()
  fmt.Printf("info: %+v\n",data)
}
```

参考答案及解析：编译失败。

```
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：

```
func main() {
  var data info

  var err error
  data.result, err = work() //ok
  if err != nil {
    fmt.Println(err)
    return
  }

  fmt.Println(data)
}
```

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?

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

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：

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

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

2.下面代码有什么问题?

```go
func main() {
  x := []int{
    1,
    2
  }
  _ = x
}
```

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：

```go
func main() {
  x := []int{    // 多行
    1,
    2,
  }
  x = x

  y := []int{3,4,} // 一行 no error
  y = y
}
```

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

3.下面代码输出什么?

```go
func test(x byte)  {
  fmt.Println(x)
}

func main() {
  var a byte = 0x11
  var b uint8 = a
  var c uint8 = a + b
  test(c)
}
```

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?

```go
func main() {
  const x = 123
  const y = 1.23
  fmt.Println(x)
}
```

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?

```go
const (
  x uint16 = 120
  y
  s = "abc"
  z
)

func main() {
  fmt.Printf("%T %v\n", y, y)
  fmt.Printf("%T %v\n", z, z)
}
```

参考答案及解析：知识点：常量。

输出：

```
uint16 120
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?

```go
func main() {
    var x string = nil

    if x == nil {
        x = "default"
    }
}
```

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：

```go
func main() {
    var x string //defaults to "" (zero value)

    if x == "" {
        x = "default"
    }
}
```

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?

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

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：

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

2.下面代码最后一行输出什么？请说明原因。

```go
func main() {
  x := 1
  fmt.Println(x)
  {
    fmt.Println(x)
    i,x := 2,2
    fmt.Println(i,x)
  }
  fmt.Println(x)  // print ?
}
```

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?

```go
type foo struct {
  bar int
}

func main() {
  var f foo
  f.bar, tmp := 1, 2
}
```

参考答案及解析：编译错误：

```
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?

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

参考答案及解析：编译错误。

```
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：

```
func main() {
  var a int8 = 3
  var b uint8 = 3
  var c int8 = -3

  fmt.Printf("^%b=%b %d\n", a, ^a, ^a) // ^11=-100 -4
  fmt.Printf("^%b=%b %d\n", b, ^b, ^b) // ^11=11111100 252
  fmt.Printf("^%b=%b %d\n", c, ^c, ^c) // ^-11=10 2
}
```

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：

```
func main() {
  var a int8 = 3
  var c int8 = 5

  fmt.Printf("a: %08b\n",a)
  fmt.Printf("c: %08b\n",c)
  fmt.Printf("a^c: %08b\n",a ^ c)
}
```

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。

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

输出：

```
x: 11010110
y: 01011100
x | y: 11011110
x &^ y: 10000010
```

第 49 天

1.下面代码输出什么?

```
func main() {
  var ch chan int
  select {
  case v, ok := <-ch:
    println(v, ok)
  default:
    println("default")
  }
}
```

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?

```
type People struct {
  name string `json:"name"`
}

func main() {
  js := `{
    "name":"seekload"
  }`
  var p People
  err := json.Unmarshal([]byte(js), &p)
  if err != nil {
    fmt.Println("err: ", err)
    return
  }
  fmt.Println(p)
}
```

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：

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

第 50 天

1.下面这段代码输出什么?

```
type T struct {
  ls []int
}

func foo(t T) {
  t.ls[0] = 100
}

func main() {
  var t = T{
    ls: []int{1, 2, 3},
  }

  foo(t)
  fmt.Println(t.ls[0])
}
```

- 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?

```
func main() {
  isMatch := func(i int) bool {
    switch(i) {
    case 1:
    case 2:
      return true
    }
    return false
  }

  fmt.Println(isMatch(1))
  fmt.Println(isMatch(2))
}
```

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：

```
func main() {
  isMatch := func(i int) bool {
    switch(i) {
    case 1:
      fallthrough
    case 2:
      return true
    }
    return false
  }

  fmt.Println(isMatch(1))     // true
  fmt.Println(isMatch(2))     // true

  match := func(i int) bool {
    switch(i) {
    case 1,2:
      return true
    }
    return false
  }

  fmt.Println(match(1))       // true
  fmt.Println(match(2))       // true
}
```

第 51 天

1.下面的代码能否正确输出?

```
func main() {
  var fn1 = func() {}
  var fn2 = func() {}

  if fn1 != fn2 {
    println("fn1 not equal fn2")
  }
}
```

参考答案及解析：编译错误

```
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?

```
type T struct {
  n int
}

func main() {
  m := make(map[int]T)
  m[0].n = 1
  fmt.Println(m[0].n)
}
```

- A. 1
- B. compilation error

参考答案及解析：B。编译错误：

```
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：

```
type T struct {
  n int
}

func main() {
  m := make(map[int]T)

  t := T{1}
  m[0] = t
  fmt.Println(m[0].n)
}
```

第 52 天

1.下面的代码有什么问题?

```go
type X struct {}

func (x *X) test()  {
  println(x)
}

func main() {

  var a *X
  a.test()

  X{}.test()
}
```

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：

```go
func main() {

  var a *X
  a.test()    // 相当于 test(nil)

  var x = X{}
  x.test()
}
```

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?

```go
func main() {
  x := map[string]string{"one":"a","two":"","three":"c"}

  if v := x["two"]; v == "" {
    fmt.Println("no entry")
  }
}
```

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：

```go
func main() {
    x := map[string]string{"one":"a","two":"","three":"c"}

    if _,ok := x["two"]; !ok {
        fmt.Println("no entry")
    }
}
```

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。

```go
type T struct {
  n int
}

func (t *T) Set(n int) {
  t.n = n
}

func getT() T {
  return T{}
}

func main() {
  getT().Set(1)
}
```

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：

```go
type T struct {
  n int
}

```

```go
func (t *T) Set(n int) {
  t.n = n
}

func getT() T {
  return T{}
}

func main() {
  t := getT()
  t.Set(2)
  fmt.Println(t.n)
}
```

第 54 天

1.下面的代码有什么问题?

```go
func (n N) value(){
  n++
  fmt.Printf("v:%p,%v\n",&n,n)
}

func (n *N) pointer(){
  *n++
  fmt.Printf("v:%p,%v\n",n,*n)
}


func main() {

  var a N = 25

  p := &a
  p1 := &p

  p1.value()
  p1.pointer()
}
```

参考答案及解析：编译错误：

```
calling method value with receiver p1 (type **N) requires explicit
dereference
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?

```
type N int

func (n N) test(){
  fmt.Println(n)
}

func main()  {
  var n N = 10
  fmt.Println(n)

  n++
  f1 := N.test
  f1(n)

  n++
  f2 := (*N).test
  f2(&n)
}
```

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：

```
func main()  {
  var n N = 10

  fmt.Println(n)

  n++
  N.test(n)

  n++
  (*N).test(&n)
}
```

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?

```
type T struct {
  n int
}

func getT() T {
  return T{}
}

func main() {
  getT().n = 1
}
```

参考答案及解析：编译错误：

```
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：

```
type T struct {
  n int
}

func getT() T {
  return T{}
}

func main() {
  t := getT()
  p := &t.n    // <=> p = &(t.n)
  *p = 1
  fmt.Println(t.n)
}
```

第 56 天

1.下面的代码有什么问题?

```go
package main

import "fmt"

func main() {
  s := make([]int, 3, 9)
  fmt.Println(len(s))
  s2 := s[4:8]
  fmt.Println(len(s2))
}
```

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?

```go
type N int

func (n N) test(){
  fmt.Println(n)
}

func main()  {
  var n N = 10
  p := &n

  n++
  f1 := n.test

  n++
  f2 := p.test

  n++
  fmt.Println(n)

  f1()
  f2()
}
```

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，能隐式第传入接收者参数。

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)
```

```
  var d data

  go func() {
    defer wg.Done()
    d.test("read")
  }()

  go func() {
    defer wg.Done()
    d.test("write")
  }()

  wg.Wait()
}
```

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：

```
func (d *data) test(s string)  {      // 指针接收者
  d.Lock()
  defer d.Unlock()

  for i:=0;i<5 ;i++  {
    fmt.Println(s,i)
    time.Sleep(time.Second)
  }
}
```

或者可以通过嵌入 *Mutex 来避免复制的问题，但需要初始化。

```
type data struct {
  *sync.Mutex      // *Mutex
}

func (d data) test(s string) {     // 值方法
  d.Lock()
  defer d.Unlock()

  for i := 0; i < 5; i++ {
    fmt.Println(s, i)
    time.Sleep(time.Second)
  }
}

func main() {

```

```
  var wg sync.WaitGroup
  wg.Add(2)

  d := data{new(sync.Mutex)}   // 初始化

  go func() {
    defer wg.Done()
    d.test("read")
  }()

  go func() {
    defer wg.Done()
    d.test("write")
  }()

  wg.Wait()
}
```

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?

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

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?

```
func main() {
  var k = 9
  for k = range []int{} {}
  fmt.Println(k)

  for k = 0; k < 3; k++ {
  }
  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++
```

```
    return n
  }
}

func main() {
  f := F(5)
  defer func() {
    fmt.Println(f())
  }()
  defer fmt.Println(f())
  i := f()
  fmt.Println(i)
}
```

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.

```
func main() {
  recover()
  panic(1)
}
```

- B.

```
func main() {
  defer recover()
  panic(1)
}
```

- C.

```
func main() {
  defer func() {
    recover()
  }()
  panic(1)
}
```

- D.

```
func main() {
  defer func() {
    defer func() {
      recover()
    }()
  }()
  panic(1)
}
```

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?

```
func main() {
  defer func() {
    fmt.Print(recover())
  }()
  defer func() {
    defer fmt.Print(recover())
    panic(1)
  }()
  defer recover()
  panic(2)
}
```

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?

```go
func main() {
  defer func() {
    fmt.Print(recover())
  }()
  defer func() {
    defer func() {
      fmt.Print(recover())
    }()
    panic(1)
  }()
  defer recover()
  panic(2)
}
```

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?

```go
type T struct {
  n int
}

func main() {
  ts := [2]T{}
  for i, t := range ts {
    switch i {
    case 0:
      t.n = 3
      ts[1].n = 9
    case 1:
      fmt.Print(t.n, " ")
    }
  }
  fmt.Print(ts)
}
```

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?

```go
type T struct {
  n int
}

func main() {
```

```
  ts := [2]T{}
  for i, t := range &ts {
    switch i {
    case 0:
      t.n = 3
      ts[1].n = 9
    case 1:
      fmt.Print(t.n, " ")
    }
  }
  fmt.Print(ts)
}
```

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?

```
type T struct {
  n int
}

func main() {
  ts := [2]T{}
  for i := range ts[:] {
    switch i {
    case 0:
      ts[1].n = 9
    case 1:
      fmt.Print(ts[i].n, " ")
    }
  }
  fmt.Print(ts)
}
```

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?

```
type T struct {
  n int
}

func main() {
  ts := [2]T{}
```

```
  for i := range ts[:] {
    switch t := &ts[i]; i {
    case 0:
      t.n = 3;
      ts[1].n = 9
    case 1:
      fmt.Print(t.n, " ")
    }
  }
  fmt.Print(ts)
}
```

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?

```
func main()  {

  for i:=0;i<10 ;i++  {
  loop:
    println(i)
  }
  goto loop
}
```

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：

```
goto loop jumps into block starting at
```

2.下面代码输出什么，请说明。

```
func main() {
  x := []int{0, 1, 2}
  y := [3]*int{}
  for i, v := range x {
    defer func() {
      print(v)
    }()
    y[i] = &v
  }
  print(*y[0], *y[1], *y[2])
}
```

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

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

- B

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

- C

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

- D

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

参考答案及解析：ACD。

2.下面代码输出什么?

```
func test(x int) (func(), func()) {
  return func() {
    println(x)
    x += 10
  }, func() {
    println(x)
  }
}

func main() {
  a, b := test(100)
  a()
  b()
}
```

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?

```
func main() {

  println(DeferTest1(1))
  println(DeferTest2(1))
}

func DeferTest1(i int) (r int) {
  r = i
  defer func() {
    r += 3
  }()
  return r
}

func DeferTest2(i int) (r int) {
  defer func() {
    r += i
  }()
  return 2
}
```

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/qiangmzsx/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。

```
type Slice []int

func NewSlice() Slice {
  return make(Slice, 0)
}
func (s *Slice) Add(elem int) *Slice {
  *s = append(*s, elem)
  fmt.Print(elem)
  return s
}
func main() {
  s := NewSlice()
  defer s.Add(1).Add(2)
  s.Add(3)
}
```

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。

```
type Slice []int

func NewSlice() Slice {
  return make(Slice, 0)
}
func (s *Slice) Add(elem int) *Slice {
  *s = append(*s, elem)
  fmt.Print(elem)
  return s
}
func main() {
  s := NewSlice()
  defer func() {
    s.Add(1).Add(2)
  }()
  s.Add(3)
}
```

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?

```
type Orange struct {
  Quantity int
}

func (o *Orange) Increase(n int) {
  o.Quantity += n
}

func (o *Orange) Decrease(n int) {
  o.Quantity -= n
}

func (o *Orange) String() string {
  return fmt.Sprintf("%#v", o.Quantity)
}

func main() {
  var orange Orange
  orange.Increase(10)
  orange.Decrease(5)
  fmt.Println(orange)
}
```

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：

```
func main() {
  orange := &Orange{}
  orange.Increase(10)
  orange.Decrease(5)
  fmt.Println(orange)
}
```

第 73 天

1.下面代码输出什么?

```
func test() []func() {
  var funs []func()
  for i := 0; i < 2; i++ {
    funs = append(funs, func() {
      println(&i, i)
    })
  }
```

```
  return funs
}

func main() {
  funs := test()
  for _, f := range funs {
    f()
  }
}
```

参考答案及解析：

```
0xc000018058 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/qiangmzsx/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?

```
var f = func(i int) {
  print("x")
}

func main() {
  f := func(i int) {
    print(i)
    if i > 0 {
      f(i - 1)
    }
  }
  f(10)
}
```

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：

```
var x = 23

func main() {
  x := 2*x - 4
  println(x)    // 输出:42
}
```

第 74 天

1.下面代码有什么问题，请说明?

```
func main() {
  runtime.GOMAXPROCS(1)

  go func() {
    for i:=0;i<10 ;i++  {
      fmt.Println(i)
    }
  }()

  for {}
}
```

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：

```
func main() {
  runtime.GOMAXPROCS(1)

  go func() {
    for i:=0;i<10 ;i++  {
      fmt.Println(i)
    }
    os.Exit(0)
  }()

  select {}
}
```

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?

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

```
func main() {
  f, err := os.Open("file")
  defer f.Close()
  if err != nil {
    return
  }

  b, err := ioutil.ReadAll(f)
  println(string(b))
}
```

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：

```
func main() {
  f, err := os.Open("file")
  if err != nil {
    return
  }
  defer f.Close()

  b, err := ioutil.ReadAll(f)
  println(string(b))
}
```

2.下面代码输出什么，为什么?

```go
func f() {
  defer func() {
    if r := recover(); r != nil {
      fmt.Printf("recover:%#v", r)
    }
  }()
  panic(1)
  panic(2)
}

func main() {
  f()
}
```

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?

```go
type S1 struct{}

func (s1 S1) f() {
  fmt.Println("S1.f()")
}
func (s1 S1) g() {
  fmt.Println("S1.g()")
}

type S2 struct {
  S1
}

func (s2 S2) f() {
  fmt.Println("S2.f()")
}

type I interface {
  f()
}

func printType(i I) {

  fmt.Printf("%T\n", i)
  if s1, ok := i.(S1); ok {
    s1.f()
    s1.g()
```

```
  }
  if s2, ok := i.(S2); ok {
    s2.f()
    s2.g()
  }
}

func main() {
  printType(S1{})
  printType(S2{})
}
```

参考答案及解析：

```
main.S1
S1.f()
S1.g()
main.S2
S2.f()
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?

```
func main() {
  var wg sync.WaitGroup
  wg.Add(1)
  go func() {
    fmt.Println("1")
    wg.Done()
    wg.Add(1)
  }()
  wg.Wait()
}
```

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?

```
func hello(num ...int) {
  num[0] = 18
}

func Test13(t *testing.T) {
  i := []int{5, 6, 7}
  hello(i...)
  fmt.Println(i[0])
}

func main() {
  t := &testing.T{}
  Test13(t)
}
```

- 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 fallthrought 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?

```go
func alwaysFalse() bool {
  return false
}

func main() {
  switch alwaysFalse()
  {
  case true:
    println(true)
  case false:
    println(false)
  }
}
```

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?

```go
type ConfigOne struct {
  Daemon string
}

func (c *ConfigOne) String() string {
  return fmt.Sprintf("print: %v", c)
}

func main() {
  c := &ConfigOne{}
  c.String()
}
```

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?

```
func main() {

  wg := sync.WaitGroup{}

  for i := 0; i < 5; i++ {
    go func(wg sync.WaitGroup, i int) {
      wg.Add(1)
      fmt.Printf("i:%d\n", i)
      wg.Done()
    }(wg, i)
  }

  wg.Wait()

  fmt.Println("exit")
}
```

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：

```
func main() {

  wg := sync.WaitGroup{}

  for i := 0; i < 5; i++ {
    wg.Add(1)
    go func(i int) {
      fmt.Printf("i:%d\n", i)
      wg.Done()
    }(i)
  }

```

```
  wg.Wait()

  fmt.Println("exit")
}
```

或者：

```
func main() {

  wg := &sync.WaitGroup{}

  for i := 0; i < 5; i++ {
    wg.Add(1)
    go func(wg *sync.WaitGroup,i int) {
      fmt.Printf("i:%d\n", i)
      wg.Done()
    }(wg,i)
  }

  wg.Wait()

  fmt.Println("exit")
}
```

第 81 天

1.下面的代码输出什么?

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

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?

```go
type TimesMatcher struct {
  base int
}

func NewTimesMatcher(base int) *TimesMatcher  {
  return &TimesMatcher{base:base}
}

func main() {
  p := NewTimesMatcher(3)
  fmt.Println(p)
}
```

- 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?

```go
func main() {
  count := 0
  for i := range [256]struct{}{} {
    m, n := byte(i), int8(i)
    if n == -n {
      count++
    }
    if m == -m {
      count++
    }
  }
  fmt.Println(count)
}
```

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?

```go
const (
  azero = iota
  aone  = iota
```

```
)

const (
  info  = "msg"
  bzero = iota
  bone  = iota
)

func main() {
  fmt.Println(azero, aone)
  fmt.Println(bzero, bone)
}
```

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。

```
type data struct {
  name string
}

func (p *data) print() {
  fmt.Println("name:", p.name)
}

type printer interface {
  print()
}

func main() {
  d1 := data{"one"}
  d1.print()

  var in printer = data{"two"}
  in.print()
}
```

参考答案及解析：编译报错。

```
cannot use data literal (type data) as type printer in assignment:
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?

```
func main() {
  a := [3]int{0, 1, 2}
  s := a[1:2]

  s[0] = 11
  s = append(s, 12)
  s = append(s, 13)
  s[0] = 21

  fmt.Println(a)
  fmt.Println(s)
}
```

参考答案及解析：

输出：

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

第 85 天

1.下面这段代码输出什么？请简要说明。

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

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?

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

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。如果想要将 src 完全拷贝至 dst，必须给 dst 分配足够的内存空间。

修复代码：

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

或者直接使用 append()

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

第 86 天

1.n 是秒数，下面代码输出什么?

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

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：

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

2.下面代码输出什么，为什么?

```
const (
  Century = 100
  Decade  = 010
  Year    = 001
)

func main() {
  fmt.Println(Century + 2*Decade + 2*Year)
}
```

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：

```
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：

```
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。

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

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：

```
0011 ^ 0010 == 0001   (3^2 == 1)
0100 ^ 0010 == 0110   (4^2 == 6)
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。

```go
func main() {
    m := make(map[string]int)
    m["foo"]++
    fmt.Println(m["foo"])
}
```

参考答案及解析：能通过编译。

上面的代码可以理解成：

```go
func main() {
    m := make(map[string]int)
    v := m["foo"]
    v++
    m["foo"] = v
    fmt.Println(m["foo"])
}
```

2.下面的代码输出什么，请简要说明?

```go
func Foo() error {
  var err *os.PathError = nil
  // …
  return err
}

func main() {
  err := Foo()
  fmt.Println(err)
  fmt.Println(err == nil)
}
```

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：

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

一个更合适的解决办法：

```go
func Foo() (err error) {
  // …
  return
}

func main() {
  err := Foo()
  fmt.Println(err)
  fmt.Println(err == nil)
}
```

第 89 天

1.下面代码能编译通过吗？请简要说明。

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

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？

```go
type P *int
type Q *int

func main() {
  var p P = new(int)
  *p += 8
  var x *int = p
  var q Q = x
  *q++
  fmt.Println(*p, *q)
}
```

- 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?

```go
type T int

func F(t T) {}

func main() {
  var q int
  F(q)
}
```

2.下面代码能通过编译吗？请简要说明。

```go
type T []int

func F(t T) {}

func main() {
    var q []int
    F(q)
}
```

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：

```
func f() {}
func f() {}

func main() {}
```

第二段：

```
func init(){}
func init(){}

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。

```
func (m map[string]string) Set(key string, value string) {
  m[key] = value
}

func main() {
  m := make(map[string]string)
  m.Set("A", "One")
}
```

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：

```go
type User map[string]string

func (m User) Set(key string, value string) {
  m[key] = value
}

func main() {
  m := make(User)
  m.Set("A", "One")
}
```

第 92 天

1.下面代码输出什么?

```go
var x int

func init() {
  x++
}

func main() {
  init()
  fmt.Println(x)
}
```

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。

```go
func min(a int, b uint) {
  var min = 0
  fmt.Printf("The min of %d and %d is %d\n", a, b, min)
}

func main() {
  min(1225, 256)
}
```

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。

```go
func min(a int, b uint) {
  var min = 0
  min = copy(make([]struct{},a),make([]struct{},b))
  fmt.Printf("The min of %d and %d is %d\n", a, b, min)
}

func main() {
  min(1225, 256)
}
```

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。

```go
type User struct {
  Name string
}

func (u *User) SetName(name string) {
  u.Name = name
  fmt.Println(u.Name)
}

type Employee User

func main() {
  employee := new(Employee)
  employee.SetName("Jack")
}
```

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。

```go
func main() {
  a := 2 ^ 15
  b := 4 ^ 15
  if a > b {
    println("a")
  } else {
    println("b")
  }
}
```

参考答案及解析：a。Go 语言里面 ^ 表示按位异或，而不是求幂。

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

2.下面哪些函数不能通过编译?

```go
func A(string string) string {
  return string + string
}

func B(len int) int {
  return len + len
}

func C(val, default string) string {
  if val == "" {
    return default
  }
  return val
}
```

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。

```go
var nil = new(int)

func main() {
  var p *int
  if p == nil {
    fmt.Println("p is nil")
  } else {
    fmt.Println("p is not nil")
  }
}
```

第 95 天

1.下面代码输出什么？请简要说明。

```go
type foo struct{ Val int }

type bar struct{ Val int }

func main() {
  a := &foo{Val: 5}
  b := &foo{Val: 5}
  c := foo{Val: 5}
  d := bar{Val: 5}
  e := bar{Val: 5}
  f := bar{Val: 5}
  fmt.Print(a == b, c == foo(d), e == f)
}
```

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?

```go
func A() int {
  time.Sleep(100 * time.Millisecond)
  return 1
}

func B() int {
  time.Sleep(1000 * time.Millisecond)
  return 2
}

func main() {
  ch := make(chan int, 1)
  go func() {
    select {
```

```
    case ch <- A():
    case ch <- B():
    default:
      ch <- 3
    }
  }()
  fmt.Println(<-ch)
}
```

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?

```
type Point struct{ x, y int }

func main() {
  s := []Point{
    {1, 2},
    {3, 4},
  }
  for _, p := range s {
    p.x, p.y = p.y, p.x
  }
  fmt.Println(s)
}
```

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：

```
type Point struct{ x, y int }

func main() {
  s := []*Point{
    &Point{1, 2},
    &Point{3, 4},
  }
  for _, p := range s {
    p.x, p.y = p.y, p.x
  }
  fmt.Println(*s[0])
  fmt.Println(*s[1])
}
```

2.下面的代码有什么隐患?

```go
func get() []byte {
  raw := make([]byte, 10000)
  fmt.Println(len(raw), cap(raw), &raw[0])
  return raw[:3]
}

func main() {
  data := get()
  fmt.Println(len(data), cap(data), &data[0])
}
```

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：

```go
func get() []byte {
  raw := make([]byte, 10000)
  fmt.Println(len(raw), cap(raw), &raw[0])
  res := make([]byte, 3)
  copy(res, raw[:3])
  return res
}

func main() {
  data := get()
  fmt.Println(len(data), cap(data), &data[0])
}
```

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。

```go
type Foo struct {
  val int
}

func (f Foo) Inc(inc int) {
  f.val += inc
}

func main() {
  var f Foo
  f.Inc(100)
  fmt.Println(f.val)
}
```

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。

```go
type Foo struct {
  val int
}

func (f *Foo) Inc(inc int) {
  f.val += inc
}

func main() {
  f := &Foo{}
  f.Inc(100)
  fmt.Println(f.val)  // 100
}
```

第 98 天

1.下面代码输出什么?

```go
func main() {
  a := 1
  for i := 0;i<5;i++ {
    a := a + 1
    a = a * 2
  }
  fmt.Println(a)
}
```

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?

```
func test(i int) (ret int) {
  ret = i * 2
  if ret > 10 {
    ret := 10
    return
  }
  return
}

func main() {
  result := test(10)
  fmt.Println(result)
}
```

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?

```
func main() {
  true := false
  fmt.Println(true)
}
```

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?

```
func watShadowDefer(i int) (ret int) {
  ret = i * 2
  if ret > 10 {
    ret := 10
    defer func() {
      ret = ret + 1
    }()
  }
  return
}

func main() {
  result := watShadowDefer(50)
  fmt.Println(result)
}
```

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?

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

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?

```go
type UserAges struct {
  ages map[string]int
  sync.Mutex
}

func (ua *UserAges) Add(name string, age int) {
  ua.Lock()
  defer ua.Unlock()
  ua.ages[name] = age
}

func (ua *UserAges) Get(name string) int {
  if age, ok := ua.ages[name]; ok {
    return age
  }
  return -1
}

func main() {
  count := 1000
  gw := sync.WaitGroup{}
  gw.Add(count * 3)
  u := UserAges{ages: map[string]int{}}
  add := func(i int) {
```

```
    u.Add(fmt.Sprintf("user_%d", i), i)
    gw.Done()
  }
  for i := 0; i < count; i++ {
    go add(i)
    go add(i)
  }
  for i := 0; i < count; i++ {
    go func(i int) {
      defer gw.Done()
      u.Get(fmt.Sprintf("user_%d", i))
    }(i)
  }
  gw.Wait()
  fmt.Println("Done")
}
```

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?

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

func generate() {
  for i := 17; i < 5000; i += 17 {
    ch <- i
    time.Sleep(1 * time.Millisecond)
  }
```

```
  close(ch)
}

func main() {
  timeout := time.After(800 * time.Millisecond)
  go generate()
  found := 0
  for {
    select {
    case i, ok := <-ch:
      if ok {
        if i%38 == 0 {
          fmt.Println(i, "is a multiple of 17 and 38")
          found++
          if found == 3 {
            break
          }
        }
      } else {
        break
      }
    case <-timeout:
      fmt.Println("timed out")
      break
    }
  }
  fmt.Println("The end")
}
```

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。

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

func generate() {
  for i := 17; i < 5000; i += 17 {
    ch <- i
    time.Sleep(1 * time.Millisecond)
  }
  close(ch)
}

func main() {
  timeout := time.After(800 * time.Millisecond)
  go generate()
  found := 0
  MAIN_LOOP:
  for {
```

```
    select {
    case i, ok := <-ch:
      if ok {
        if i%38 == 0 {
          fmt.Println(i, "is a multiple of 17 and 38")
          found++
          if found == 3 {
            break MAIN_LOOP
          }
        }
      } else {
        break MAIN_LOOP
      }
    case <-timeout:
      fmt.Println("timed out")
      break MAIN_LOOP
    }
  }
  fmt.Println("The end")
}
```

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。

```
var mu sync.Mutex
var chain string

func main() {
  chain = "main"
  A()
  fmt.Println(chain)
}
func A() {
  mu.Lock()
  defer mu.Unlock()
  chain = chain + " --> A"
  B()
}

func B() {
  chain = chain + " --> B"
```

```
  C()
}

func C() {
  mu.Lock()
  defer mu.Unlock()
  chain = chain + " --> C"
}
```

- 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?

```
func main() {
  fmt.Println(doubleScore(0))
  fmt.Println(doubleScore(20.0))
  fmt.Println(doubleScore(50.0))
}
func doubleScore(source float32) (score float32) {
  defer func() {
    if score < 1 || score >= 100 {
      score = source
    }
  }()
  return source * 2
}
```

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。

```
var mu sync.RWMutex
var count int

func main() {
  go A()
  time.Sleep(2 * time.Second)
  mu.Lock()
```

```
  defer mu.Unlock()
  count++
  fmt.Println(count)
}
func A() {
  mu.RLock()
  defer mu.RUnlock()
  B()
}
func B() {
  time.Sleep(5 * time.Second)
  C()
}
func C() {
  mu.RLock()
  defer mu.RUnlock()
}
```

- 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。

```go
func main() {
  var wg sync.WaitGroup
  wg.Add(1)
  go func() {
    time.Sleep(time.Millisecond)
    wg.Done()
    wg.Add(1)
  }()
  wg.Wait()
}
```

- 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。

```go
var c = make(chan int)
var a int

func f() {
  a = 1
  <-c
}
func main() {
  go f()
  c <- 0
  print(a)
}
```

- 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。

```go
type MyMutex struct {
  count int
  sync.Mutex
}
```

```
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() {
```

```
  var ch chan int
  go func() {
    ch = make(chan int, 1)
    ch <- 1
  }()
  go func(ch chan int) {
    time.Sleep(time.Second)
    <-ch
  }(ch)

  c := time.Tick(1 * time.Second)
  for range c {
    fmt.Printf("#goroutines: %d\n", runtime.NumGoroutine())
  }
}
```

- 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 groutine 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。

```
func main() {
  var m sync.Map
  m.LoadOrStore("a", 1)
  m.Delete("a")
  fmt.Println(m.Len())
}
```

- 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。

```go
func main() {
  var wg sync.WaitGroup
  wg.Add(2)
  var ints = make([]int, 0, 1000)
  go func() {
    for i := 0; i < 1000; i++ {
      ints = append(ints, i)
    }
    wg.Done()
  }()
  go func() {
    for i := 0; i < 1000; i++ {
      ints = append(ints, i)
    }
    wg.Done()
  }()
  wg.Wait()
  fmt.Println(len(ints))
}
```

- 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?

```go
type People struct {
  name string `json:"name"`
}

func main() {
  js := `{
    "name":"11"
  }`
  var p People
  err := json.Unmarshal([]byte(js), &p)
  if err != nil {
    fmt.Println("err: ", err)
    return
  }
  fmt.Println("people: ", p)
}
```

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 prot 参数，默认值是 0.0.0.0 和 8000。

```
var ip string
var port int

func init() {
  // A
  // B
}

func main() {
  flag.Parse()
  fmt.Printf("%s:%d", ip, port)
}
```

参考答案及解析：flag 包的使用。

```
var ip string
var port int

func init() {
  flag.StringVar(&ip, "ip", "0.0.0.0", "ip address")
  flag.IntVar(&port, "port", 8000, "port number")
}

func main() {
  flag.Parse()
  fmt.Printf("%s:%d", ip, port)
}
```

第 109 天

1.下面代码有什么问题?

```
func main() {
  ch := make(chan int, 1000)
  go func() {
    for i := 0; i < 10; i++ {
      ch <- i
    }
  }()
  go func() {
    for {
```

```
        a, ok := <-ch
        if !ok {
          fmt.Println("close")
          return
        }
        fmt.Println("a: ", a)
      }
    }()
    close(ch)
    fmt.Println("ok")
    time.Sleep(time.Second * 20)
}
```

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。

```
type S struct {
  v int
}

func main() {
  s := []S{{1}, {3}, {5}, {2}}
  // A
  fmt.Printf("%#v", s)
}
```

参考答案及解析：可以考虑使用 sort.Slice()。

```
type S struct {
  v int
}

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

第 110 天

1.下面代码输出什么？请简要说明。

```
type T struct {
  V int
```

```
}

func (t *T) Incr(wg *sync.WaitGroup) {
  t.V++
  wg.Done()
}
func (t *T) Print() {
  time.Sleep(1)
  fmt.Print(t.V)
}
func main() {
  var wg sync.WaitGroup
  wg.Add(10)
  var ts = make([]T, 10)
  for i := 0; i < 10; i++ {
    ts[i] = T{i}
  }
  for _, t := range ts {
    go t.Incr(&wg)
  }
  wg.Wait()
  for _, t := range ts {
    go t.Print()
  }
  time.Sleep(5 * time.Second)
}
```

- 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。

```
const N = 26

func main() {
  const GOMAXPROCS = 1
  runtime.GOMAXPROCS(GOMAXPROCS)

  var wg sync.WaitGroup
  wg.Add(2 * N)
  for i := 0; i < N; i++ {
```

```
    go func(i int) {
      defer wg.Done()
      // A
      runtime.Gosched()
      fmt.Printf("%c", 'a'+i)
    }(i)
    go func(i int) {
      defer wg.Done()
      fmt.Printf("%c", 'A'+i)
    }(i)
  }
  wg.Wait()
}
```

参考答案及解析：

```
const N = 26

func main() {
  const GOMAXPROCS = 1
  runtime.GOMAXPROCS(GOMAXPROCS)

  var wg sync.WaitGroup
  wg.Add(2 * N)
  for i := 0; i < N; i++ {
    go func(i int) {
      defer wg.Done()
      runtime.Gosched()
      fmt.Printf("%c", 'a'+i)
    }(i)
    go func(i int) {
      defer wg.Done()
      fmt.Printf("%c", 'A'+i)
    }(i)
  }
  wg.Wait()
}
```

第 111 天

1.下面两处打印的值是否相同？请简要说明。

```go
func main() {
  var val int
  println(&val)
  f(10000)
  println(&val)
}

func f(i int) {
  if i--; i == 0 {
    return
  }
  f(i)
}
```

参考答案及解析：不同。知识点：栈增长、逃逸分析。每个 groutine 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。

```go
func main() {
  var val int

  a := &val
  println(a)

  f(10000)

  b := &val
  println(b)

  println(a == b)  // A
}

func f(i int) {
  if i--; i == 0 {
    return
  }
  f(i)
}
```

- A. ture
- 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。

```
func main() {
  var val int

  a := &val
  println(a)

  f(10000)

  b := &val
  println(a)   // a b 的值相同
  println(b)

  println(a == b) // A
}

func f(i int) {
  if i--; i == 0 {
    return
  }
  f(i)
}
```

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。

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

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。

```
func main() {

  x := make([]int, 0, 7)
  x = append(x, 100, 200, 300, 400, 500, 600, 700)
  twohundred := &x[1]
  x = append(x, 800)
  for i := range x {
    x[i]++
  }
  fmt.Println(*twohundred)    // 输出 200

  x = make([]int, 0, 8)   // 指向另一个切片
  x = append(x, 100, 200, 300, 400, 500, 600, 700)
  twohundred = &x[1]
  x = append(x, 800)  // 执行 append 操作，容量足够，不会重新申请内存
  for i := range x {
    x[i]++
  }
  fmt.Println(*twohundred)  // 输出 201
}

```

2.下面的代码输出什么？请简要说明。

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

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.

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

B.

```
const (
  size int64= 1024
  eof = -1
)
```

C.

```
const (
  ERR_ELEM_EXISTerror = errors.New("element already exists")
  ERR_ELEM_NT_EXISTerror = errors.New("element not exists")
)
```

D.

```
const u, vfloat32 = 0, 3
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] 。

```
func link(p ...interface{}) {
  fmt.Println(p)
}

func main() {
  link("seek", 1, 2, 3, 4) // 输出 [seek 1 2 3 4]
  a := []int{1, 2, 3, 4}
  link("seek", a) // 输出 [seek [1 2 3 4]]
}
```

参考答案及解析：

```
func link(p ...interface{}) {
  fmt.Println(p)
}

func main() {
  link("seek", 1, 2, 3, 4) // 输出 [seek 1 2 3 4]
  a := []int{1, 2, 3, 4}
  link("seek", a) // 输出 [seek [1 2 3 4]]

  tmplink := make([]interface{}, 0, len(a)+1)
  tmplink = append(tmplink, "seek")
  for _, ii := range a {
    tmplink = append(tmplink, ii)
  }
```

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

第 114 天

1.下面代码输出什么?

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

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。

```
func main() {
  i := 0
  f := func() int {
    i++
    return i
  }
  c := make(chan int, 1)
  c <- f()
  select {
  case c <- f():
  default:
    fmt.Println(i)
  }
}
```

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：

```
func main() {
  i := 0
  f := func() int {
    fmt.Println("incr")
    i++
    return i
  }
  c := make(chan int)
  for j := 0; j < 2; j++ {
    select {
    case c <- f():
      // noop
```

```
    default:
      // noop
    }
  }
  fmt.Println(i)
}
```

第 115 天

1.下面正确的是?

```
var y int

func f(x int) int {
  return 7
}

A.
switch y = f(2) {
case y == 7:
  return
}

B.
switch y = f(2); {
case y == 7:
  return
}

C.
switch y = f(2) {
case 7:
  return
}

D.
switch y = f(2); {
case 7:
  return
}
```

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?

```go
func main() {
  a := []int{1, 2, 3, 4}
  b := variadic(a...)
  b[0], b[1] = b[1], b[0]
  fmt.Println(a)
}

func variadic(ints ...int) []int {
  return ints
}
```

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?

```go
const (
    one = 1 << iota
    two
)

func main() {
    fmt.Println(one, two)
}
```

2.下面的代码输出什么?

```go
const (
  greeting = "Hello, Go"
  one = 1 << iota
  two
)

func main() {
  fmt.Println(one, two)
}
```

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?

```
const N = 10

func main() {
  m := make(map[int]int)

  wg := &sync.WaitGroup{}
  mu := &sync.Mutex{}
  wg.Add(N)
  for i := 0; i < N; i++ {
    go func() {
      defer wg.Done()
      mu.Lock()
      m[i] = i
      mu.Unlock()
    }()
  }
  wg.Wait()
  println(len(m))
}
```

参考答案及解析：输出 1。知识点：并发、引用。修复代码如下：

```
const N = 10

func main() {
  m := make(map[int]int)

  wg := &sync.WaitGroup{}
  mu := &sync.Mutex{}
  wg.Add(N)
  for i := 0; i < N; i++ {
    go func(i int) {
      defer wg.Done()
      mu.Lock()
      m[i] = i
      mu.Unlock()
    }(i)
  }
  wg.Wait()
  println(len(m))
}
```

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?

```
const (
    _ = iota
    c1 int = (10*iota)
    c2
    d = iota
)
func main() {
  fmt.Printf("%d - %d - %d",c1,c2, d)
}
```

- 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/Iota

第 119 天

1.关于slice或map操作，下面正确的是。

A.

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

B.

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

C.

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

D.

```
var m map[string]int
m = make(map[string]int)
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。

```
var ErrDidNotWork = errors.New("did not work")

func DoTheThing(reallyDoIt bool) (err error) {
  if reallyDoIt {
    result, err := tryTheThing()
    if err != nil || result != "it worked" {
      err = ErrDidNotWork
    }
  }
  return err
}

func tryTheThing() (string, error) {
  return "", ErrDidNotWork
}

func main() {
  fmt.Println(DoTheThing(true))
  fmt.Println(DoTheThing(false))
}
```

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：

```go
func DoTheThing(reallyDoIt bool) (err error) {
  var result string
  if reallyDoIt {
    result, err = tryTheThing()
    if err != nil || result != "it worked" {
      err = ErrDidNotWork
    }
  }
  return err
}
```

第 120 天

1.下面代码输出什么?

```go
func main() {
  fmt.Println(len("你好bj!"))
}
```

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?

```go
func GetValue(m map[int]string, id int) (string, bool) {
  if _, exist := m[id]; exist {
    return "存在数据", true
  }
  return nil, false
}

func main() {

  intmap := map[int]string{
    1: "a",
    2: "bb",
    3: "ccc",
  }

  v, err := GetValue(intmap, 3)
  fmt.Println(v, err)
}
```

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，但是运行是时候会报:cannot use nil as type string in return argument.

引自《Go夜读》