函数

language 不能失去它的function，就像泥车不能没有它的兵

声明

基础声明

函数声明包括函数名、形式参数列表、返回值列表（可省略）以及函数体

```
func name(para-list) (return-list) {
    body
}
```

就像

```
func WinAndWin(win string) (string, string) {
    declare(win)
    return win, win
}
```

返回值可以省略？怎么省？🤔
没有，不久省了吗😋

```
void printWin() {
    cout<<"win\n";
}
```

```
func printWin() { // 不用非得来个void了
    fmt.Println("win")
}
```

参数列表

```
func f(i, j, k int, s, t string)                 { /* ... */ }
func f(i int, j int, k int,  s string, t string) { /* ... */ }
```

是一样的

```
int a, b, c = int a; int b; int c;
```

命名返回值

我们返回值也要有名字!
有名字 == 可操作

```
func add(a, b int) (res int) {
        res = a + b
        return // 等价于return a + b
}
```

函数签名

函数的"类型"

```
func add(x int, y int) int   {return x + y}
func sub(x, y int) (z int)   { z = x - y; return}
func first(x int, _ int) int { return x } // _ blank identifier 强调某个参数未被使用
func zero(int, int) int      { return 0 } // 我只知道传进来两个int，是什么不知道😋

fmt.Printf("%T\n", add)   // "func(int, int) int"
fmt.Printf("%T\n", sub)   // "func(int, int) int"
fmt.Printf("%T\n", first) // "func(int, int) int"
fmt.Printf("%T\n", zero)  // "func(int, int) int"
```

No More Default

Go 中没有默认实参

```cpp
// in C++ √
int add(int a = 0, int b = 0) {
    return a+b;
}

// in Go ×
func add(a int = 0, b int = 0) int {
    return a+b
}
```

替代方案?
e.g. 可变参数

```go
func add(nums ...int) int { // ...表示该参数可以接受任意数量的该类型值
    sum := 0
    for _, num := range nums { // 本质是可变参数在函数内被转换为一个切片Slice
        sum += num
    }
    return sum
}

// 调用
fmt.Println(add())       // 0
fmt.Println(add(1))      // 1
fmt.Println(add(1, 2))   // 3
```

递归

函数调用自身

e.g. visit()

```
func visit(links []string, n *html.Node) []string {
        ...
        for c := n.FirstChild; c != nil; c = c.NextSibling {
                links = visit(links, c) // visit递归调用visit
        }
        return links
}
```

怎么跑?

`getLinks()` 函数命令行传参方法

1. 直接输入，Ctrl-Z结束输入

```
# go run 运行
go run funcs.go
# 复制粘贴到终端中
<!DOCTYPE html>
<html>
<body>
    <a href="https://example.com">Example</a>
</body>
</html>
# 按 Ctrl+C 结束
```

2. 通过管道传递HTML

```
echo '<a href="https://example.com">Example</a>' | go run funcs.go
```

3. 从文件读取HTML
   假设文件 `test.html`

```
<!DOCTYPE html>
<html>
<body>
    <a href="https://example.com">Example</a>
    <a href="https://google.com">Google</a>
</body>
</html>
```

在cmd里运行

```
go run findlinks1.go < test.html
```

在powershell里运行

```
Get-Content test.html | go run funcs.go
```

多返回值

详见ch2/multiReturn哦~ 😘

错误

why error?

举个例子，任何进行I/O操作的函数都会面临出现错误的可能，只有没有经验的程序员才会相信读写操作不会失败，即使是简单的读写——《Go语言圣经》

错误是重要组成部分，error是正常结果之一

错误处理！没问题就OK，有问题要throw / report

了解更多的错误信息——error 类型

`error == nil` ，说明没出错，反正则出错

错误处理策略

传播错误

某个子程序失败，整个函数失败(return)

```
resp, err := http.Get(url)
if err != nil{ // 出错了
    return nil, err
}
```

重新尝试

e.g. 连接失败，再尝试重连（一般有限时长/次数）

```
// WaitForServer attempts to contact the server of a URL.
// It tries for one minute using exponential back-off.
// It reports an error if all attempts fail.
func WaitForServer(url string) error {
    const timeout = 1 * time.Minute
    deadline := time.Now().Add(timeout)
    for tries := 0; time.Now().Before(deadline); tries++ { // deadline前重复尝试连接（如果出现错误
        _, err := http.Head(url)
        if err == nil {
            return nil // success
        }
        log.Printf("server not responding (%s);retrying…", err)
        time.Sleep(time.Second << uint(tries)) // exponential back-off
    }
    return fmt.Errorf("server %s failed to respond after %s", url, timeout)
}
```

输出错误信息并结束程序(Exit)

should be only in `main()`

```
// (In function main.)
if err := WaitForServer(url); err != nil {
    fmt.Fprintf(os.Stderr, "Site is down: %v\n", err) // report
    os.Exit(1) // and exit
}


if err := WaitForServer(url); err != nil {
    log.Fatalf("Site is down: %v\n", err)
}
```

效果于上面的相同，log库更简洁

只需要输出错误信息

```
print(error)
```

```
if err := Ping(); err != nil {
    log.Printf("ping failed: %v; networking disabled",err)
}
```

或者标准错误流输出错误信息

```
if err := Ping(); err != nil {
    fmt.Fprintf(os.Stderr, "ping failed: %v; networking disabled\n", err)
}
```

直接忽略

不愧是最后一招呀，Go学长这招太狠了😇 😋

What's more, 检查某个子函数是否失败后，我们通常将处理失败的逻辑代码放在处理成功的代码之前

```
// 先处理失败
if error != nil {
    // 处理错误
}

// 正常操作
```

函数值

《第一类值》

一等公民！函数也有颗变量心

函数像其他值一样，拥有类型，可以被赋值给其他变量，传递给函数，从函数返回

```
func add(a, b int) int {
    return a + b
}

func main() {
    // 将函数赋值给变量
    // 类型：func(int, int) int (in 函数签名)
    var sum func(int, int) int = add
    // sum := add 当然也OK

    // 使用变量调用函数
    result := sum(3, 5)
    fmt.Println(result) // 输出：8
}
```

lambda函数

C/C++：都说了，借鉴记得标明出处！ 😋

匿名函数，也称闭包(closure)

```
add := func(a, b int) int {
    return a+b
}

c := add(1, 2)
```

可以和C++ lambda一样花

```
// 排序
people := []Person{
        {"Alice", 25},
        {"Bob", 30},
        {"Charlie", 20},
    }

// 按年龄排序
// sort.Slice(slice[], cmp())
sort.Slice(people, func(i, j int) bool {
    return people[i].Age < people[j].Age
})
```

递归调用

没有名字的函数怎么调用自己呢？🤔
很简单，起个名字就是咯！😋

```
// 先声明一个函数变量
var factorial func(int) int

// 然后定义匿名函数并赋值
factorial = func(n int) int {
    if n <= 1 {
        return 1
    }
    return n * factorial(n-1) // 递归调用
}

fmt.Println(factorial(5)) // 输出: 120
```

这样太不麻烦了，不学不学

哎，我这还有使用闭包实现递归（Y 组合子风格）和固定点组合子（Y Combinator）实现

怕吓到你，你还是去求那个8字（母）仙人问个明白吧🤫

高级用法

函数作为参数

```
func calculate(a, b int, operation func(int, int) int) int {
    return operation(a, b)
}

func add(a, b int) int {
    return a + b
}

sum := calculate(1, 2, add) // 3
```

函数作为返回值

```
// 函数造函数
func doubleWin(win string) func(string) string {
    return func(win string) string {
        return win + win
    }
}

func main() {
    win := "win"
    winAndWin := doubleWin(win) // 返回的winAndWin是一个函数！
    fmt.Println(winAndWin(win))
}
```

你们一等公民玩的就是花😋